Volume 22 Number 3

ACTA CYBERNETICA

Szeged, 2016

Helmut Jürgensen
Department of Computer Science
Middlesex College
The University of Western Ontario
London, Canada
hjj@csd.uwo.ca

Zoltan Kato
Department of Image Processing
and Computer Graphics
Szeged, Hungary
kato@inf.u-szeged.hu

Alice Kelemenová
Institute of Computer Science
Silesian University at Opava
Opava, Czech Republic
Alica.Kelemenova@fpf.slu.cz

László Lovász
Department of Computer Science
Eötvös Loránd University
Budapest, Hungary
lovasz@cs.elte.hu

Gheorghe Păun
Institute of Mathematics of the
Romanian Academy
Bucharest, Romania
George.Paun@imar.ro

András Prékopa
Department of Operations Research
Eötvös Loránd University
Budapest, Hungary
prekopa@cs.elte.hu

Arto Salomaa
Department of Mathematics
University of Turku
Turku, Finland
asalomaa@utu.fi

László Varga
Department of Software Technology
and Methodology
Eötvös Loránd University
Budapest, Hungary
varga@ludens.elte.hu

Heiko Vogler
Department of Computer Science
Dresden University of Technology
Dresden, Germany
Heiko.Vogler@tu-dresden.de

Gerhard J. Woeginger
Department of Mathematics and
Computer Science
Eindhoven University of Technology
Eindhoven, The Netherlands
gwoegi@win.tue.nl

Acta Cybernetica **22** (2016) 553–571.

Identifying Code Clones with RefactorErl*

Viktória Fördős† and Melinda Tóth‡

Abstract

Code clones, the results of "copy&paste programming", have a negative impact on software maintenance. Therefore several tools and techniques have been developed to identify them in the source code. Most of them concentrate on imperative, well known languages, while in this paper, we give an AST/metric based clone detection algorithm for the functional programming language Erlang. We propose a standalone solution that does not overload users with results that are insignificant from the point of view of the user. We emphasise that the maintenance costs can be decreased by using our solution, because the programmers need to deal only with important issues.

1 Introduction

Duplicated code detectors [3, 7] help in the identification of clones. Various approaches [23] have been proposed, including the analysis of code tokens [17], the syntax tree built up using the tokens [22], and using different metrics [22]. The majority of these methods and algorithms have been constructed specifically for the imperative paradigm and its mainstream languages, whilst in functional programming only a few exist, such as [6] developed for the Haskell language, and [11, 15] for the Erlang language [2]. Hitherto, none of the published papers dealt with the issue of *irrelevant clones* and proposed a standalone solution to the problem of presenting only the *relevant clones* to the user.

In practice, the set of (initial) clones is too large and contains many *false positive* or *irrelevant clones*. Therefore further operations are needed to narrow down the result set to serve the user only valuable, relevant clones as results.

First of all, we should discuss the difference between *false positive* and irrelevant clones. False positive clones are not real clones, whilst irrelevant clones are real clones, but they are absolutely useless. Examples of both kinds of clones are shown in Figure 1.

Clone detection algorithms focus only on false positive clones during filtering, and do not usually deal with irrelevant clones. Our goal is to construct an algorithm which easily notices important clones. So, we have tried to filter out any

*Supported by Ericsson–ELTE-Soft–ELTE Software Technology Lab

†ELTE-Soft Ltd., E-mail: f-viktoria@elte.hu

‡Eötvös Loránd University, E-mail: tothmelinda@elte.hu

DOI: 10.14232/actacyb.22.3.2016.1

False positive clone	Irrelevant clone
`f(List) -> 1+length(List).` `g() -> self() ! message.`	`new_cg() -> #callgraph{}.` `new_plt() -> #plt{}.`

Figure 1: Examples of false positive and irrelevant clones

clones serving no useful purpose. We have observed that the complete result of the algorithm can be ruined by a huge amount of irrelevant clones, because the users are not capable of distinguishing important clones from irrelevant ones while they are being swamped with worthless details. Our filtering system is the second phase of Clone IdentifiErl, and is detailed in Section 4.4.

Clone detection is a special static analysis task and its precision is hugely influenced by the available information, therefore Clone IdentifiErl does not work directly on the source code.

RefactorErl [1, 5, 29] is a static source code analyser and transformer tool for Erlang. RefactorErl provides a representation that contains even more information about the source beyond that of the abstract syntax tree.

Contributions In this paper we introduce Clone IdentifiErl that is an AST/metric based algorithm, whose implementation exploits the advantages of RefactorErl to precisely detect clones in Erlang programs. We address the problem of serving only relevant clones as results by proposing an Erlang specific solution. Moreover, we compare our algorithm with other Erlang specific detectors and we discuss how this solution can also be tailored to efficiently deal with the typical irrelevant clones of other programming languages.

2 Related work

The clone research community has carried out significant research for the last two decades. Various clone detection approaches have been proposed. The simplest algorithm is the line-based detection [23], where the recurrences of source code lines are detected. Although the most commonly used techniques are token and syntax based methods [3, 4, 20], some approaches build a sequence database from the source code and use fingerprints for the detection of clones [24, 25]. Mayrand et al. [22] use a metric based approach to identify code clones. Mohammad et al. [19] dealt with clones that are active at runtime to determine the impact of these clones.

Only a few researchers dealt with the problem of irrelevant clones. Harsu et al. [14] have published a case study classifying the importance of clones. Jurgens et al. [16] have proposed an iterative, configurable clone detector, called ConQAT, that contains a filtering system. ConQAT can remove repetitive generated code fragments and overlapping clones by iteratively reconfiguring and rerunning its initial clone detector. Contrary to ConQAT, our approach is a standalone filtering system,

thus it can be plugged into an initial clone detector and necessitates neither iterative evaluation nor the reproduction of initial clones to filter out irrelevant clones. We have proposed a more flexible, language-independent filtering system [12] to fit the preferences of any user. This filtering system works with groups of clones to refine them based on the domain specific predicates given by the user.

Some research has been carried out to ease the comprehension of the result of duplicated code detection. Here, a key is the compactness of the result. But a duplicated code detector can only result in pairs of clones or groups of clones. The latter scenario is said to be more comprehensible. Although, if the representation of the algorithm does not aid in retrieving grouped clones, grouping the result is a further step. SeClone [18] supports automatic grouping on file-level type usage by using the Suffix Tree Clustering algorithm. Tairas et al. [27] use Latent Semantic Indexes (LSI) to group clone classes that are the result of a syntax-driven clone detection algorithm. In general, LSI uses the singular value decomposition technique to identify relations between terms and concepts in unstructured text. The proposed approach exploits LSI to reveal relationships among clone classes that are not based on syntactical structures. We have also proposed a solution [10] addressing this problem that can be used almost in any cases when the result consists of clones pairs.

3 Erlang and RefactorErl

Erlang is a declarative, dynamically typed, functional, concurrent programming language, which was designed to develop soft real-time, distributed applications.

The compilation unit of Erlang programs is called a *module*, which is built up from attributes and function definitions. The encapsulating module, the name of the function, and the arity of the function can identify a function uniquely in Erlang. Pattern matching features are a prominent way to define functions by cases. The cases of a function definition are called *function clauses*, and they are separated from each other by a ; token. A one-arity function, which consists of two function clauses, is shown in Erlang source 1. This function will be our running example through out the paper.

A function clause is built up from either one expression, called the *top-level expression*, or a sequence of top-level expressions as defined in the Erlang grammar. There are no statements in Erlang, only expressions. Contrary to statements, every expression has a value, which is the value of its last top-level expression.

Two kinds of expressions, the *list comprehension* and the *record* expression, can be found in the implementation of almost all industrial applications. Thus our filtering system focuses on them, and we briefly introduce these expressions here.

List is a frequently used data structure in Erlang. A list comprehension is a built-in language feature of Erlang to manipulate a list, based on Zermelo-Fraenkel set theory [13]. A general list comprehension is shown in Erlang source 2, whose *generators* (`Gen1`, ... ,`GenN`) are responsible for producing the base set. The *filters* (`Filter1`, ... `FilterN`) of the list comprehension narrow the base sets. `Expr`,

```
clone_fun(L) when is_list(L)->
    ShortVar = L,
    A = 1,
    B = lists:max([I || I<-lists:seq(1, 10)]),
    (A == 1) andalso throw(badarg),
    self ! B;

clone_fun(_)->
    V = f(g(42)),
    LongVariableName = V,
    B = lists:max([J || J<-lists:seq(V, V*2)]),
    X = fun(E) -> E + B end,
    self ! X.
```

Erlang source 1: clone_fun/1 function definition form

```
[ Expr || Pattern1<-Gen1, Filter1, ... ,  PatternN<-GenN, FilterN ]
```

Erlang source 2: A general list comprehension

called a *head of the list comprehension*, is an expression, which is evaluated on every element of the generators for which all filters are true.

Due to its simplified and safe usage, a record is an important preprocessed language element of Erlang, which is similar to a struct in C. There are four kinds of record operations:

- gathering the index of a record field is a non-modifier record operation;
- accessing a value of a field is a non-modifier record operation;
- creating an instance of a record is a modifier record operation;
- modifying a value of a field is a modifier record operation.

RefactorErl supports the daily work of Erlang programmers with code comprehension and refactoring tools. It provides the ability to retrieve semantic information and metric values about the source code, to perform dependency analysis and to visualise the results of the analysis. It facilitates code reorganisation with clustering algorithms and several refactoring methods. The incremental and asynchronous analyser architecture allows the programmer to track source code changes. The tool has multiple user interfaces to choose from: a web-based interface, an interactive console or one can use Emacs or Vim with RefactorErl plugins.

The source code has to be loaded into RefactorErl in order to be analysed. While performing analyses, the tool builds a labelled, directed graph, called *Semantic Program Graph* containing lexical, syntactic and semantic information about the source code. Information from the Semantic Program Graph is gathered by the evaluation of path expressions and the traversal of the graph. The algorithm presented in this paper does use information from the Semantic Program Graph and metrics

of RefactorErl.

4 Clone IdentifiErl

In this chapter, we present a new algorithm for accurate clone detection. Our algorithm combines a number of existing techniques, but introduces also a novel filtering component, to be described in Section 4.4. To our knowledge, these techniques have never been used specifically in Erlang.

What does clone detection mean intuitively? One may try to compare every code fragment to every other. The original representation of a code fragment is too concrete, thus a generalised form of source code needs to be used. The similarity of each pair of code fragments can be represented by a matrix. The first component of our algorithm produces this matrix, which is detailed in Section 4.2. From this matrix, the *initial clones* can be extracted along diagonals. This is what the second component of our algorithm does, which is described in Section 4.3. Irrelevant clones can be found among these clones, which are removed by evaluating filters. This process is described in Section 4.4.

4.1 Unit

The unit of a clone instance has to be chosen as cautiously as possible. One of our goals was to design and construct an algorithm that can be successfully used on legacy code, so the source code of several Erlang programs were studied.

The abstraction level of Erlang is high. Due to this abstraction, an application written in Erlang is so brief that a line of Erlang code generally corresponds to 8 to 10 lines of C code. It follows that block-based algorithms cannot be used. It also follows that the size of the chosen unit should be small. Tokens and subexpressions are too small to be used efficiently and a function clause is not small enough, therefore a top-level expression becomes the unit of the algorithm.

The program text of a top-level expression is too particular, thus generalisation is needed. We convert the expressions into a formal language, which uses a formal alphabet. This formal language can hide the unneeded specialisations of the tokens.

A generalised top-level expression is a sentence over the fixed formal alphabet. Every word is produced based on the type of the token. Tokens are produced by tokenizing expressions in the same order as given by the lexical analyser. It is necessary to preserve this order to keep the characteristics of the original expression. The alphabet of the language is not injective, in order to hide unneeded differences, for example, the difference between a variable and a constant (either a number or an atom).

Example After generalisation, our running example will be as shown in Figure 2. All top-level expressions are indexed and generalised.

Index	Top-level expression	Generalised top-level expr.
	`clone_fun(L) when is_list(L)->`	
i-1	`ShortVar = L,`	`A=A`
i	`A = 1,`	`A=A`
i+1	`B = lists:max([I || I<-lists:seq(1, 10)]),`	`A=A:A([AlAvA:A(A,A)])`
i+2	`(A == 1) andalso throw(badarg),`	`(AfA)FA(A)`
i+3	`self ! B;`	`A!A`
	`clone_fun(_)->`	
j-1	`V = f(g(42)),`	`A=A(A(A))`
j	`LongVariableName = V,`	`A=A`
j+1	`B = lists:max([J || J<-lists:seq(V, V*2)]),`	`A=A:A([AlAvA:A(A,A*A)])`
j+2	`X = fun(E) -> E + B end,`	`A=x(A)zA+Ae`
j+3	`self ! X.`	`A!A`

Figure 2: The transformation part of the first component

4.2 Matrix

A code clone is usually a result of "copy&paste programming". As an example, assume that one has copied a three-unit long sequence and has modified the second unit of the sequence, but the order of the sequence has been kept unchanged.

Usually larger clones are preferred, so we want to collect the three-unit long sequence as one clone instead of collecting three one-unit long clones. To be able to do it, modifications should be handled flexibly. Our algorithm works primarily on a matrix, which is a view of the problem, with which the flexibility criteria can be satisfied. Each element of the matrix expresses the similarity between two expressions and while a clone is made by preserving the original, correct order of its elements, it is enough to focus on the diagonals of a matrix. In other words, the fragments of diagonals are completely isomorphic to the fragments of code sequences found in the code directly. We put this idea in perspective in the following subsections.

4.2.1 Introducing the matrix

Assume that every top-level expression is numbered (indexed) sequentially, as shown in Figure 2. By taking the cardinality of the indexes as the size (denoted by n), a square matrix can be constructed, whose elements express similarity between the defining rows and columns, which are the top-level expressions identified by their indexes.

The relation, denoted by *Similarity*, between two top-level expressions, has the following properties:

- *Similarity* is reflexive, namely all values are related to themselves.

	1	$i-1$	i	$i+1$	$i+2$	$i+3$	n
1	...	...	...	...	...	...	...
$j-1$	⋮	0.5	0.5	0.43	0.46	0	⋮
j	⋮	**1.0**	**1.0**	0.21	0	0	⋮
$j+1$	⋮	0.19	0.19	**0.94**	0.23	0	⋮
$j+2$	⋮	0.17	0.17	0.22	0.24	0	⋮
$j+3$	⋮	0	0	0	0	**1.0**	⋮
n	...	...	...	...	...	...	...

Figure 3: A Dice-Sørensen similarity matrix

- *Similarity* is symmetric.
- *Similarity* quantifies the similarity between two top-level expressions in a clear and distinctive manner.

If the symmetric property holds, then only the lower triangular matrix need to be computed. If the reflexive property also holds, it follows that the elements of the main diagonal do not need to be computed. With these two properties the volume of computation is greatly reduced to the following cardinality:

$$\tfrac{1}{2}n^2 - n.$$

Clone IdentifiErl uses Dice-Sørensen metric [8, 26] for determining similarity, which does satisfy the properties of *Similarity* relation, too. There is no reason why the metric should not be replaced with other string similarity metrics [28]. Let Dice-Sørensen metric be portrayed by the m function

$$m : String \times String \to [0, 1] \subset \mathbb{R}.$$

Let n be the cardinality of the top-level expressions, A be the n-sized, square matrix. Let *selecttle* be a selector function which returns the top-level expression indexed by the given index. Now the matrix can be defined as

$$A(i,j) ::= \begin{cases} m(selecttle(i), selecttle(j)) & \text{if } i,j \in [1,n], i < j; \\ 0 & \text{otherwise.} \end{cases}$$

Example Consider the code fragments shown in Figure 2 with indexes. The relevant part of the Dice-Sørensen similarity matrix is shown in Figure 3.

4.2.2 Patterns in the matrix

The clauses are clones of each other, except that line `(i-1)` differs from line `(j-1)` and line `(i+2)` also greatly differs from line `(j+2)`. Therefore, it can be said that

Index	Top-level expression	Generalised top-level expr.
	`clone_fun(L) when is_list(L)->`	
i-1	`ShortVar = L,`	`A=A`
i	`A = 1,`	`A=A`
i+1	`B = lists:max([I || I<-lists:seq(1, 10)]),`	`A=A:A([AlAvA:A(A,A)])`
i+2	`(A == 1) andalso throw(badarg),`	`(AfA)FA(A)`
i+3	`self ! B;`	`A!A`
	`clone_fun(_)->`	
j-1	`V = f(g(42)),`	`A=A(A(A))`
j	`LongVariableName = V,`	`A=A`
j+1	`B = lists:max([J || J<-lists:seq(V, V*2)]),`	`A=A:A([AlAvA:A(A,A*A)])`
j+2	`X = fun(E) -> E + B end,`	`A=x(A)zA+Ae`
j+3	`Y = lists:zip([1,2,3],[3,21]),`	`A=A:A([A,A,A],[A,A,A])`
j+4	`self ! X.`	`A!A`

Figure 4: The new definition of `clone_fun/1`

three clones are present: the first one is a one-unit long pair, namely `([i-1], [j])`, the second one is also a one-unit long pair, namely `([i+3], [j+3])`, and the third one is a two-unit long pair, namely `([i, i+1], [j, j+1])`.

The following pairs are related to each other according to relation *isClone* (which is formally defined in section 4.3):

$$\{\ldots,(i-1,j),(i,j),(i+1,j+1),(i+3,j+3),\ldots\} = isClone.$$

Thanks to the complexity of Erlang programs one-unit long clone pairs can still be relevant clones. However, multi-unit long clone pairs are preferred in practice.

To take another example, assume that the starting units of a k-unit long clone pair can be found at indices a and b (k is a positive, fixed integer). Then

$$\{(a+i,b+i) \mid i \in [0\ldots k-1] \subset \mathbb{Z}\} \subseteq isClone.$$

As observed by Baker [3], every pair in the defined set is an element of the matrix, and based on a k-unit long clone pair one of the diagonals of the matrix can be partially formed.

There may exist clones that cannot be found among diagonals such as the following. Let us assume that the first clause of `clone_fun/1` is the same as shown in Figure 2, but its second clause contains one newly inserted top-level expression. The new definition of `clone_fun/1` is shown in Figure 4.

Clone pairs `([i+3],[j+4])` and `([i, i+1], [j, j+1])` are in different diagonals. If the instances of a clone differ from each other in that way, then the full clone cannot be collected from the same diagonal, for instance, when the cardinality of inserted, deleted or rewritten top-level expressions differ from each other.

To summarise, instead of finding any pattern in the matrix, it is enough to search in diagonals. Although a full clone cannot be collected from the same diagonal in every case, its parts can be collected from different diagonals.

4.3 Determining initial clones

While a clone may be divided into sub clones due to insertions, deletions or other kinds of modifications, it would be practical if a full clone could be gathered somehow. Therefore we need to add a new parameter, called the *invalid sequence length.* It is the maximum length of a sequence whose middle elements can differ more from each other than *threshold* would allow. This limitation to the elements is naturally needed because of the beginnings and the endings of the clones should be similar to each other. By introducing invalid sequence length, one can customise the allowable maximum deviation of a clone.

If the chosen metric is exactly a distance, its values should be normalised to $[0, 1]$ to be able to handle the threshold correctly.

Now, we are able to precisely define the *isClone* relation, which expresses whether two units are considered to be clones of each other. The Dice-Sørensen metric is portrayed by the m function, and *Threshold* contains a non-negative real number that is less than one. Let *isClone* be a general Boolean function operating on string pairs as follows:

$$isClone : String \times String \to \mathbb{L}.$$

The truth set of this function is:

$$[isClone] ::= \{(a, b) \mid a \in String, b \in String, m(a, b) > Threshold\}.$$

As shown in Section 4.2, it is enough to focus only on the diagonals. Thus, if the set of diagonals is constructed first, the elements of the set can be computed in parallel, because every element of the matrix is affected by only one complete diagonal.

We calculate the initial clones [9] by traversing the diagonals in parallel. We check whether a pair of top-level expressions is a clone or not. In the former case we try to extend the initial clone candidate with a new pair of expressions. In the latter case we take into account the invalid sequence length and try to extend the candidate if it is allowed.

Working with diagonals has a deficiency: the gathered instances of a clone can overlap the natural boundaries of the clone. The overlap should be avoided if possible. So, a boundary needs to be defined as a trimming rule of the production of initial clones, as follows: every top-level expression of a clone must belong to the same function clause per instance. This rule works, because function clauses act like natural boundaries.

Example The three initial clones which are detected by the described algorithm with using 1 for invalid sequence length are shown below:

1. `LongVariableName = Var`
   and
   `ShortVar = L`

2. ```
   A = 1,
   B = lists:max([I || I<-lists:seq(1, 10)]),
   (A == 1) andalso throw(badarg),
   self ! B
   ```
   and
   ```
   LongVariableName = Var,
   B = lists:max([J || J<-lists:seq(Var, Var*2)]),
   X = fun(E) -> E + B end,
   self ! X
   ```

3. `A = 1`
   and
   `ShortVar = L`

4.4 Filtering and trimming unit

There is an important difference between one-unit long and multi-unit long clones. Due to the high abstraction level of the formal language used to tokenise the program text, and the usage of the similarity metric, lots of false positive and irrelevant clones appear in the set of initial clones if only the one-unit long clones are taken into consideration. It follows that the filters for one-unit long clones need to be stricter than the filters for the multi-unit long clones.

As described in Section 4.3, the first phase of the algorithm exploits the advantages of invalid sequence length to point out clones that cannot be found by some of the other algorithms, for instance, the algorithm [15] that primarily works on a suffix tree to gather the initial clones. This asset should be preserved, thus invalid sequence length is also used in the filtering unit to process the multi-unit long clones. During the filtering, it can happen that a multi-unit long clone is split into a one-unit long clone and the rest of the multi-unit long clone. In this case, the one-unit long clone has to be further processed by the filters that are relevant for one-unit long clones.

4.4.1 Algorithm of the filtering system

A clone appears in the result set of the algorithm only if it meets all the requirements which are stated in the corresponding filters. For all *clone* in $InitialClones$, we have:

$$\bigwedge_{Filter \in Filters} Filter(clone) \implies clone \in ResultClones.$$

Crucially, if there is a requirement (defined by one of the filters) cannot be fulfilled, the clone is dropped. That makes our filtering system easily extendible and also very efficient, because the evaluation of filters is short-circuit.
   ```

The basic idea behind the algorithm is shown in Algorithm 1, which is detailed in Algorithm 2 by focusing only on the multi-unit long clones. If the currently examined clone is a one-unit long clone, then the `FiltersForOneLongs` function is responsible for dealing with it. The `FiltersForOneLongs` function forms the conjunction of the results of the evaluated filters, which are dedicated to one-unit long clones. If the conjunction is true, then the examined clone is returned, otherwise an empty set is returned.

```
function FilteringAndTrimmingUnit(InitialClones, InvSeqLength)
    Clones ← ∅
    parallel for all Clone ∈ InitialClones do
        if isOneUnitLongClone(Clone) then
            Clones ← FiltersForOneLongs(Clone)
        else
            Clones ← FiltersForMultiLongs(Clone, InvSeqLength)
        end if
    end parallel for
    return Clones
end function
```

Algorithm 1: Filtering and trimming unit of the algorithm

The input of the algorithm is the set of the initial clones and the invalid sequence length. The execution can run in parallel on initial clones, which are given as input to Algorithm 1. Two scenarios can occur after applying the filtering system to a one-unit long clone pair: it can be accepted or removed from the final result. Applying the filtering system to a multi-unit long clone pair can result in multiple smaller, to-be-filtered clone pairs. Nevertheless to any scenario the algorithm terminates in a finite number of steps. The output of the algorithm is a set of clones, which are produced in parallel, so the result of the algorithm is comprised of the separate result sets.

A bit more explanation is needed for the `FurtherTrim` function. This function is responsible for trimming invalid items from the beginnings and endings of the given clone. The result of a trimming operation is a set, whose one-unit long clones are further filtered by the `FiltersForOneLongs` function, to check if the stricter filters, defined to deal with one-unit long clones, can be satisfied by these clones, too.

4.4.2 Description of the defined filters

In this subsection the filters are presented. They can be categorised into three groups. The first group is used on one-unit long clones, the second group deals with multi-unit long clones, and the third group is applied to every clone. The ideas behind the filters were based on separate case studies on the detected initial clones of a real life application, called Mnesia [21]. Mnesia, written in Erlang, is a database management system and belongs to the standard Erlang/OTP library. It

```
function FiltersForMultiLongs(Clone, InvSeqLength)
    Clones ← ∅
    AClone ← ⟨⟩
    InvSeqCount ← 0
    for all UnitPair ∈ Clone do
        if ∧FilterFun∈FilterFuns* FilterFun(UnitPair) then
            AClone ← AClone ⊕ ⟨UnitPair⟩
            InvSeqCount ← 0
        else
            if InvSeqCount < InvSeqLength then
                AClone ← AClone ⊕ ⟨UnitPair⟩
                InvSeqCount ← InvSeqCount + 1
            else
                Clones ← Clones ∪ FurtherTrim(AClone)
                AClone ← ⟨⟩
                InvSeqCount ← 0
            end if
        end if
    end for
    Clones ← Clones ∪ FurtherTrim(AClone)
    return Clones
end function
```

Algorithm 2: Filtering and trimming unit of the multi-unit long clones

consists of 22,594 non-empty lines of code, 31 modules, 1,687 functions and 5,393 top-level expressions. Thanks to our filtering system, all irrelevant clones, which are shown as examples of filters in this subsection, appear only in the set of initial clones found in Mnesia and are not present in the final result set.

Filters for one-unit long clones As stated earlier, stricter filters need to be used on one-unit long clones, because a huge amount of these clones are in fact useless.

The *Simple Expression filter* is responsible for dropping a pair of top-level expressions if at least one component of the pair is an atom, a number, a character, a variable, a list, a tuple, a record operation or a function application. It may seem too strict, but in practice nobody cares about clones like the following ones: `Res` and `false`.

The *Simple Match filter* is responsible for dropping a pair of top-level expressions if only one component of the pair is a match expression, or if both are match expressions with right-hand sides that would be dropped by the Simple Expression filter, or if both components are match expressions with right-hand sides whose referred functions differ from each other. An example can be the following pair: `true = ets:foldl(Insert, true, Tab)` and `DelObjs = mnesia_lib:db_get(Tab, K)`.

The *Simple Send filter* is responsible for dropping a pair of top-level expressions

if only one component of the pair is a send expression or if both components are send expressions with right-hand sides that would be dropped by the Simple Expression filter. For example, consider `ReplyTo ! {self(),Done}` and `Pid ! {self(), more, Slot}`.

Filters for multi-unit long clones First, notice that a clone is examined here in such a way that all filters are separately evaluated on all its units (see Algorithm 2) that are pairs of top-level expressions. If a pair exists that cannot satisfy all the filters, then the clone is trimmed further. While studying filters for multi-unit long clones, this notice should be kept in mind.

The longer a clone is, the more important it is. Thus, the constraints of the filters for multi-unit long clones need to be weaker and also more flexible than the constraints of the filters for one-unit long clones. Therefore, we have introduced a new filter type, called *compound filter*, which satisfies the previously stated requirements. A filter is a compound filter if it is evaluated in the following way. If the kind of the examined expression is a match expression or a send expression, then instead of the given expression, its right-hand side is taken into consideration. The algorithm benefits from using compound filters while dealing with multi-unit long clones, because these filters find the dominant part of an expression, so the evaluation of the filter shows relevant result.

The *Same Right-hand Side filter* is responsible for dropping a pair of top-level expressions if only one component of the pair is a send or match expression, or both components are send or match expressions with right-hand sides of the following kinds: atom, number, character, variable or string, or if both components are match expressions with right-hand sides whose referred functions differ from each other. For instance, consider `Dist2 = incr_node(Node, Dist)` and `Tab = element(1, Val)`.

The *Same Record Operation filter* – a compound filter – is responsible for dropping a pair of top-level expressions if only one component of the pair is a record operation or if both components are record operations which differ in the classification of the operation (either a modifier or a non-modifier) or differ in the fields of the referred record. An example is as follows: `Tid = D#decision.tid` and `Commit0 = P#participant.commit`.

Filters for any clones The following two filters focus on branching expressions or fun expressions, which are built up from clauses and are frequently used in Erlang.

The *Same Cardinality filter* – a compound filter – is responsible for dropping a pair of top-level expressions if only one component of the pair is a branching or fun expression or if both components are branching or fun expressions which differ in the cardinality of the clauses. As an example consider Figure 5.

The *Same Function Application filter* – a compound filter – is responsible for dropping a pair of top-level expressions if only one component of the pair is a branching or fun expression or if both components are branching or fun expressions

```
case Res of                    case Res of
    {'EXIT', Reason} ->            {'EXIT', {aborted, What}} -> abort(What);
        exit(Reason);              {'EXIT', What} -> abort(What);
    _ -> Res                       _ ->Res
end                            end
```

Figure 5: An example dropped by the Same Cardinality filter

which differ in the function calls considering at least one of its clause. For example, `Fun = fun(S) -> lists:suffix(S, File) end` and `Fun2 = fun(Frag, A) -> mnesia:foldr(ActivityId, Opaque, Fun, A, Frag, LockKind) end`.

The *Head of List Comprehension filter* – a compound filter – is responsible for dropping a pair of top-level expressions if only one component of the pair is a list comprehension, or if both components are list comprehensions and their head expressions differ either in the cardinality of their sub-expressions or in referred functions. For instance, consider `Dist = [{good, Node, 0} || Node <- Pool]` and `TableList = [{Tab, keep_tables} || Tab <- List]`.

Example From the three initial clones gathered from our running example, only the four-unit long clone is the result of the algorithm, the two one-unit long clones are filtered out. These clones are object lessons for irrelevant clones.

To demonstrate the necessity of both the filtering system and the stricter filters for one-unit long clones, one should consider that 346,130 one-unit long and 5,022 multi-unit long initial clones were found in Mnesia. The set of the initial clones can greatly be narrowed down by using all defined filters, thus the result of the algorithm is more comprehensible in practice. As a demonstration of the efficiency of the filtering system, consider that 351,152 initial clones were found in Mnesia, and this huge amount of initial clones was reduced to 801 by applying the filtering system. Neither irrelevant nor false positive clones were found in the result of the algorithm. A non-trivial example is shown in Figure 6.

We end this section by describing how Clone IdentifiErl should be advanced to detect clones in programs written in another programming languages. The first phase requires a parser that decomposes the analysed program into small syntactic units. These units should be tokenised to turn them into their generalised representatives by using a formal language. The remaining phases of the initial clone detection will use these generalised forms. The second phase requires more effort. At first, the categories of the irrelevant clones need to be determined with which the necessary filters can be characterised. To materialise these filters, it is almost certain that a static analyser need to be employed. Next, the filtering phase should be implemented by using the same technique as we have presented in this paper but by using the proper filters.

```
Left one (found in mnesia_loader): case ?catch_val(send_compressed) of
 {'EXIT', _} -> mnesia_lib:set(send_compressed, NoCompression),
 NoCompression; Val -> Val end

Right one (found in mnesia_controller):
    case ?catch_val(no_table_loaders) of
        {'EXIT', _} ->
            mnesia_lib:set(no_table_loaders,1),
            1;
        Val -> Val
    end
```

Figure 6: Clones from the `mnesia` library

5 Comparison with another Erlang specific algorithms

We have examined separate programs (Mnesia and the clustering application of RefactorErl) by using Clone IdentifiErl, our metric-based clone detector and a standard suffix-tree based approach. In this chapter, we discuss the lessons we learnt.

The result sets of the algorithms are slightly different. The difference originates from the separate theoretical backgrounds of the algorithms, as we will discuss in this chapter.

Regardless of the applied theories, real syntactic clones are reported by all algorithms. This is obvious if we consider that the instances of these clones are similar from any point of view. However, even if syntactic clones can be reported by all the algorithms, the metric-driven algorithm can overlook some clones that are smaller than a function.

Considering the differences between Clone IdentifiErl and the suffix-tree based algorithm, the differences originate from independent reasons. Generally speaking, Clone IdentifiErl is more powerful; it results in more clones than the suffix-tree based algorithm does.

Clone IdentifiErl is not restricted by the parameter that constraints clones to consists of at least a minimum number of tokens. Thus clones that are smaller than this minimum do not appear in the result of the suffix-tree based algorithm.

Due to the usage of string similarity metrics, Clone IdentifiErl can find not only real syntactic clones. This is a great advantage that allows the users to detect clones whose instances differ from each other because of small modifications. For instance, consider Erlang Source 3.

As described previously, the theories driving the algorithms greatly influence their results. All the algorithms work, but serve separate purposes. If those clones are in the focus that can easily be eliminated, then the usage of the suffix-tree based algorithm is advised because of its linear computational cost. If the constraints are

```
% First instance
sorensen_dice2(_E1, Attr1, _E2, Attr2) ->
 #presence{both=A,first=B,second=C,none=D}=presence(Attr1,Attr2),
 if
A+B+C+D == 0 -> 1;
true         -> 1 - 2*(A+D) / (2*A + B + C + D)
 end.

% Second instance
sorensen_dice(_E1, Attr1, _E2, Attr2) ->
 #presence{both=A, first=B, second=C} = presence(Attr1, Attr2),
 if
A+B+C == 0 -> 1;
true       -> 1 - 2*A / (2*A + B + C)
 end.
```

Erlang source 3: A clone can only be detected by using string similarity metrics

weakened and clones that are modified instances of each other are also concerned, then Clone IdentifiErl can come to the rescue. But if only clones that are functions are important, the metric-driven algorithm is the ideal candidate. We additionally note that the metric-driven algorithm is more broadly usable than it seems to be at first sight, because functions in Erlang are small; functions usually consist of a few (one to five) top-level expressions.

6 Conclusion and future works

Duplicated code detection is a special static analysis, where code clones are identified in the source code. Clones can result in several bugs and inconsistencies during software maintenance. Thus, programmers should at least be aware of their existence.

In this paper we have described and evaluated a duplicated code detection algorithm to identify code clones in Erlang programs. We have shown the main parts of Clone IdentifiErl by highlighting the filtering possibilities. We use the representation of Erlang programs defined by RefactorErl (a static analyser and transformer tool) to build the matrix and to evaluate filters.

We have discussed the problem of presenting only valuable clones to the user. We have learnt that the complete result of a duplicated code detector can be ruined if both irrelevant and relevant clones are presented. By taking the initial clones of Mnesia into consideration, we have demonstrated how serious this problem can be if the clones of a legacy code base need to be identified.

We have proposed an Erlang specific solution – the filtering system – which narrows down the set of initial clones by filtering out both irrelevant and false

positive clones. By applying the filtering system, Clone IdentifiErl can distinguish between irrelevant and relevant clones to provide a more comprehensible result to the users. To best of our knowledge, no paper have proposed a standalone solution to filter out irrelevant clones.

We want to further evaluate and improve our techniques. We are going to further study the results of our approach and tune the algorithm by altering the number of used filters.

References

[1] RefactorErl Homepage. http://refactorerl.com.

[2] Armstrong, Joe. *Programming Erlang: Software for a Concurrent World.* Pragmatic Bookshelf, 2007.

[3] Baker, Brenda S. A program for identifying duplicated code. In *Computer Science and Statistics: Proc. Symp. on the Interface*, pages 49–57, March 1992.

[4] Baxter, Ira D., Yahin, Andrew, Moura, Leonardo, Sant'Anna, Marcelo, and Bier, Lorraine. Clone Detection Using Abstract Syntax Trees. In *Proceedings of the International Conference on Software Maintenance*, ICSM '98, pages 368–377. IEEE Computer Society, 1998.

[5] Bozó, I., Horpácsi, D., Horváth, Z., Kitlei, R., Kőszegi, J., Tejfel, M., and Tóth, M. RefactorErl - Source Code Analysis and Refactoring in Erlang. In *Proceedings of the 12th Symposium on Programming Languages and Software Tools*, pages 138–148, Tallin, Estonia, October 2011.

[6] Brown, Christopher and Thompson, Simon. Clone Detection and Elimination for Haskell. In Gallagher, John and Voigtlander, Janis, editors, *PEPM'10: Proceedings of the 2010 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, pages 111–120, January 2010.

[7] Cordy, James R. and Roy, Chanchal K. The NiCad Clone Detector. In *Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension*, ICPC '11, pages 219–220. IEEE Computer Society, 2011.

[8] Dice, Lee Raymond. Measures of the Amount of Ecologic Association Between Species. *Ecology*, 26(3):297–302, July 1945.

[9] Fördős, Viktória and Tóth, Melinda. Identifying Code Clones with RefactorErl. In *Proceedings of the 13th Symposium on Programming Languages and Software Tools*, pages 31–45, Szeged, Hungary, August 2013.

[10] Fördős, Viktória and Tóth, Melinda. Comprehensible presentation of clone detection results. In *To appear in AIP Conference Proceedings of the 12th International Conference of Numerical Analysis and Applied Mathematics*, Rhodes, Greece, September 2014.

[11] Fördős, Viktória and Tóth, Melinda. Utilising the software metrics of RefactorErl to identify code clones in Erlang. In *Proceedings of 10th Joint Conference on Mathematics and Computer Science*, volume LIX of *Informatica*, pages 103–118, Cluj-Napoca, Romania, May 2014. Studia Universitatis Babeş-Bolyai.

[12] Fördős, Viktória, Tóth, Melinda, and Kozsik, Tamás. Clone Wars. In *Proceedings of the 3th Workshop on Software Quality Analysis, Monitoring, Improvement and Applications*, volume 1266, pages 15–22, Lovran, Croatia, September 2014.

[13] Fraenkel, A.A. and Hillel, Y.B. *Foundations of Set Theory.* Foundations of mathematics. North-Holland Publishing Company, 1958.

[14] Harsu, Maarit, Bakota, Tibor, Siket, István, Koskimies, Kai, and Systä, Tarja. Code Clones: Good, Bad, or Ugly? In *Proceedings of 11th Symposium on Programming Languages and Software Tools and 7th Nordic Workshop on Model Driven Software Engineering*, pages 162–176, Tampere, Finland, August 2009.

[15] Huiqing, Li and Simon, Thompson. Clone detection and removal for Erlang/OTP within a refactoring environment. In *Proceedings of the 2009 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation*, PEPM '09, pages 169–178. ACM, 2009.

[16] Juergens, Elmar and Göde, Nils. Achieving Accurate Clone Detection Results. In *Proceedings of the 4th International Workshop on Software Clones*, IWSC '10, pages 1–8, New York, NY, USA, 2010. ACM.

[17] Kamiya, T., Kusumoto, S., and Inoue, K. CCFinder: a multilinguistic token-based code clone detection system for large scale source code. *IEEE Transactions on Software Engineering*, 28(7):654–670, 2002.

[18] Keivanloo, I., Rilling, J., and Charland, P. SeClone - A Hybrid Approach to Internet-Scale Real-Time Code Clone Search. In *Program Comprehension (ICPC), 2011 IEEE 19th International Conference*, pages 223–224, June 2011.

[19] Khan, Mohammad Asif A., Schneider, Kevin A, and Roy, Chanchal K. Active Clones: Source Code Clones at Runtime. In *Proceedings of the Eighth International Workshop on Software Clones*, volume 63 of *IWSC'14*, July 2014.

[20] Koschke, Rainer and Riemann, Ole Jan Lars. Robust Parsing of Cloned Token Sequences. In *Proceedings of the Eighth International Workshop on Software Clones*, volume 63 of *IWSC'14*, July 2014.

[21] Mattsson, Haakan, Nilsson, Hans, and Wikstrom, Claes. Mnesia - A Distributed Robust DBMS for Telecommunications Applications. In *PADL '99: Proceedings of the First International Workshop on Practical Aspects of Declarative Languages*, pages 152–163. Springer-Verlag, 1998.

[22] Mayrand, Jean, Leblanc, Claude, and Merlo, Ettore. Experiment on the Automatic Detection of Function Clones in a Software System Using Metrics. In *Proceedings of the 1996 International Conference on Software Maintenance*, ICSM '96, pages 244–. IEEE Computer Society, 1996.

[23] Roy, Chanchal K., Cordy, James R., and Koschke, Rainer. Comparison and Evaluation of Code Clone Detection Techniques and Tools: A Qualitative Approach. *Sci. Comput. Program.*, 74(7):470–495, May 2009.

[24] Schleimer, Saul, Wilkerson, Daniel S., and Aiken, Alex. Winnowing: local algorithms for document fingerprinting. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 76–85. ACM, 2003.

[25] Smith, Randy and Horwitz, Susan. Detecting and Measuring Similarity in Code Clones. In *3rd REF/TCSE International Workshop on Software Clones. Workshop proceedings of the 13th European Conference on Software Maintenance and Reengineering*, CSMR2009, pages 28–34, Kaiserslautern, Germany, March 2009. IEEE Computer Society.

[26] Sørensen, T. A method of establishing groups of equal amplitude in plant sociology based on similarity of species and its application to analyses of the vegetation on Danish commons. *Biol. Skr.*, 5:1–34, 1948.

[27] Tairas, Robert and Gray, Jeff. An Information Retrieval Process to Aid in the Analysis of Code Clones. volume 14, pages 33–56, Hingham, MA, USA, February 2009. Kluwer Academic Publishers.

[28] Thierry, Lavoie and Ettore, Merlo. About Metrics for Clone Detection. In *Proceedings of the Eighth International Workshop on Software Clones*, volume 63 of *IWSC'14*, July 2014.

[29] Tóth, Melinda and Bozó, István. Static analysis of complex software systems implemented in Erlang. In *Proceedings of the 4th Summer School Conference on Central European Functional Programming School*, CEFP'11, pages 440–498, Berlin, Heidelberg, 2012. Springer-Verlag.

Received 10th March 2014

Acta Cybernetica **22** (2016) 573–589.

Improving Saturation-based Bounded Model Checking*

Dániel Darvas[†] András Vörös[†] and Tamás Bartha[‡]

Abstract

Formal verification is becoming a fundamental step in assuring the correctness of safety-critical systems. Since these systems are often asynchronous and even distributed, their verification requires methods that can deal with huge or even infinite state spaces. Model checking is one of the current techniques to analyse the behaviour of systems, as part of the verification process.

In this paper a symbolic bounded model checking algorithm is presented that relies on efficient saturation-based methods. The previous approaches are extended with new bounded state space exploration strategies. In addition, constrained saturation is also introduced to improve the efficiency of bounded model checking. Our measurements confirm that these approaches do not only offer a solution to deal with infinite state spaces, but in many cases they even outperform the original methods.

Keywords: model checking, symbolic model checking, CTL, bounded model checking, saturation, asynchronous systems, compacting saturation

1 Introduction

The usage of formal methods is one of the possible solutions to assure the quality of the more and more complex safety-critical, embedded systems. For this reason, highly efficient formal verification algorithms are needed. *Model checking* is one of the most prevalent formal verification technique. It is an automatic approach to check whether the formal model (and thus the modelled system) satisfies its specification. This specification can be expressed e.g. in *Computation Tree Logic* (CTL) that is a popular temporal logic language thanks to the efficient verification algorithms.

*This work was partially supported by the ARTEMIS JU and the Hungarian Ministry of National Development (NFM) in frame of the R5-COP (Reconfigurable ROS-based Resilient Reasoning Robotic Cooperating Systems) project.

[†]Department of Measurement and Information Systems, Budapest University of Technology and Economics, Budapest, Hungary. E-mail: darvas@mit.bme.hu, vori@mit.bme.hu.

[‡]Institute for Computer Science and Control, MTA SZTAKI, Budapest, Hungary.

DOI: 10.14232/actacyb.22.3.2016.2

Model checking traverses the state space of the model being analysed. Safety-critical systems are often asynchronous, even distributed, therefore the composite state space of their asynchronous subsystems can be as large as the Cartesian product of the local components' state spaces, i.e., the state space of the whole system "explodes". *Symbolic methods* [7] are advanced techniques to handle huge state spaces of synchronous systems: instead of storing states explicitly, symbolic techniques rely on an encoded representation of the state space. Ordinary symbolic methods, however, usually perform poorly for asynchronous systems.

Saturation [3] is considered as one of the most effective state space generation and model checking algorithms for asynchronous systems. It combines the efficiency of symbolic methods with a special iteration strategy. Saturation-based state space exploration computes the set of reachable states. The so-called *saturation-based structural model checking* algorithm can evaluate temporal logic properties. Nowadays, the so-called *constrained saturation-based structural model checking* algorithm is one of the most efficient algorithms for CTL model checking [13].

However, the state space of many complex models is either infinite, or too large to be represented even symbolically. In these cases *bounded model checking* can be a solution, as it explores and examines the prescribed properties on a bounded part of the state space. *Bounded saturation-based state space exploration* was introduced in [12], i.e., an algorithm that explores the state space only to some bounded depth.

Motivation and Previous Work. Former saturation-based approaches solved only one of the problems: they could either be used for structural model checking over the entire state space; or they could traverse the state space up to a given bound, but without checking complex properties. In the method presented here these two approaches are integrated. Our algorithm incrementally explores the state space and performs structural model checking on the discovered bounded partial state space. Furthermore, bounded model checkers usually do not support the full CTL [8]. Although there were theoretical results in this field, former bounded model checking approaches did not work well with CTL due to its branching semantics.

This paper extends our former work [11] described in Sect. 3.1 with new state space exploration strategies and with an efficient formula evaluation algorithm (namely constrained saturation) to traverse the bounded state space. This is the first approach where the constrained saturation is used in bounded model checking.

The structure of our paper is as follows: Sect. 2 introduces the background and prerequisites of our work. Sect. 3 gives an overview of the advanced saturation-based algorithms our work relies on. Sect. 4 describes the improvements of the bounded CTL model checking algorithm. Sect. 5 presents our measurement results. At the end, our conclusions and ideas for future work complete the paper.

2 Background

In this section we outline the theoretical background of our work. First, we describe *decision diagrams*, which form the underlying data structures of our algorithms.

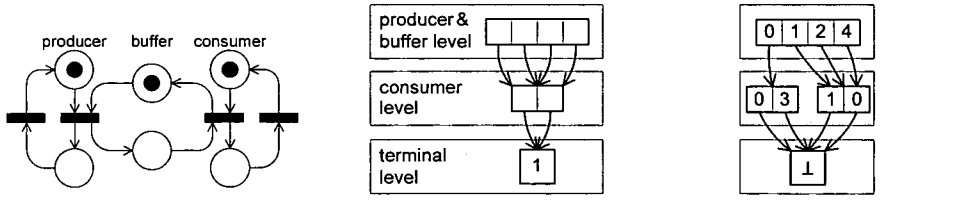
After that, we introduce the principles of bounded model checking. Finally, we overview the saturation-based state space exploration algorithm and the model checking background.

2.1 Decision Diagrams

This section is based on [11]. Decision diagrams are used in symbolic model checking for efficiently storing the state space and the possible state changes of the models. A *Multiple-valued Decision Diagram* (MDD) [3] is a directed acyclic graph, representing a function f consisting of K variables: $f : \{0, 1, \ldots\}^K \to \{0, 1\}$. An MDD has a node set containing two types of nodes: non-terminal nodes and terminal nodes (terminal 0 and terminal 1). The nodes are ordered into $K + 1$ levels. All *non-terminal nodes* are labelled by a variable index $1 \leq k \leq K$, which indicates the level the node belongs to (the variable it represents), and has n_k (domain size of the variable) arcs pointing to nodes in level $k - 1$. The *terminal nodes* are labelled by the variable index 0.

In our representation, duplicate nodes are not allowed, so if two nodes have identical successors in level k, they are also identical. These rules ensure that MDDs provide a canonical and compact representation of a given function or set. The function is evaluated by traversing the MDD top-down via the variable assignments represented by the arcs between nodes.

Figure 1(a) depicts a simple example Petri net [9] model of a producer-consumer system. The producer creates items and places them in the buffer, from where the consumer removes them. For synchronizing purposes, the buffer's capacity is one, so the producer has to wait till the consumer takes away the item from the buffer. This Petri net model has a finite state space containing 8 states. Figure 1(b) depicts an MDD used for storing the encoded state space. Each edge encodes a possible local state [3]. The possible (global) states are the paths from the root node to the terminal *one* node. (The model has to be decomposed to be able to represent its state space using decision diagrams efficiently. This *decomposition* will be discussed in Section 2.3.)



(a) The Petri net of the producer-consumer model

(b) State space representation with MDD

(c) State space and distance representation with EDD

Figure 1: Producer-consumer example

The bounded saturation-based state space exploration algorithms use *Edge-valued Decision Diagrams* (EDDs) too. EDDs extend MDDs with extra informa-

tion: they can represent $f : \{0, 1, \ldots\}^K \to \mathbb{N} \cup \{\infty\}$ functions. It means that in addition to storing the state space, they can also store the distance information for bounded state space generation. As it is not directly used in our contributions, we omit the detailed introduction of EDDs and we refer the reader to [12] for details.

2.2 Model Checking and Bounded Model Checking

Given a formal model, *model checking* [7] is an automated technique to decide whether the model satisfies the specification. Formally: let M be a Kripke structure and let f be a formula of temporal logic. The goal of model checking is to find all states s of a model M such that $M, s \vDash f$ (meaning that the state s of model M satisfies f).

Bounded model checking [2, 6] decides whether the model satisfies the specification in a predefined number of steps, i.e., the depth of the state space traversal. Formally the problem for the k-bounded state space is to find all states s of M such that $M, s \vDash_k f$ (meaning that the state s of model M satisfies f in at most k steps). Among others, bounded model checking is useful when the full state space is not needed to decide on a property. This is e.g. the case for *shallow bugs.*

Structural model checking uses set operations to evaluate temporal logic specifications by computing fixed points in the state space. CTL [7] is a widely used temporal logic specifications formalism. It has expressive syntax, and structural model checking provides efficient algorithms to analyse CTL specifications. CTL expressions contain state variables, Boolean operators, and *temporal operators.* Temporal operators occur in pairs in CTL: the path quantifier, either A (on all paths) or E (there exists a path), is followed by the tense operator, one of X (next), F (future, or finally), G (globally), and U (until). However, only three: EX, EU, EG of the 8 possible pairings is enough to express all the possible expressions due to duality [7].

2.3 Saturation

Saturation is a *symbolic algorithm* for state space generation and model checking. *Decomposition* serves as the prerequisite for the symbolic encoding: the algorithm maps the state variables of the chosen high-level formalism into symbolic variables of the decision diagram. The global state of the model can be represented as the composition of the local states of components: $\mathsf{s} = (s_1, s_2, \ldots, s_n)$, where n is the number of components. See Figure 1(b) for a possible decomposition and the corresponding MDD representation of the example model in Figure 1(a). Furthermore, decomposition helps the algorithm to efficiently *exploit locality*, which is inherent in asynchronous systems. Locality ensures that a transition usually affects only a limited set of components or just certain parts of the submodels. The algorithm divides the global next state function $\mathcal{N}$ into smaller parts: $\mathcal{N} = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e$, where $\mathcal{E}$ is the set of events in the high level model. The granularity of the decomposition can be chosen arbitrarily [5].

Saturation uses *symbolic encoding of the next state function.* In our work we use the symbolic next state representation from [1]. We represent $\mathcal{N}$ as a set of state

pairs, thus it can be stored in a decision diagram. For this set representation $\mathcal{R}$, the following is true: $\mathbf{s}' \in \mathcal{N}(\mathbf{s}) \Leftrightarrow (\mathbf{s},\mathbf{s}') \in \mathcal{R} \Leftrightarrow \mathcal{R}(\mathbf{s},\mathbf{s}')$ *is true.* The global relation can be expressed by the symbolic next state relations of the events: $\mathcal{R}(\mathbf{s},\mathbf{s}') = \bigvee_{e\in\mathcal{E}} \mathcal{R}_e(\mathbf{s},\mathbf{s}')$. Applying $\mathcal{N}_e$ to a given set of states represented by S results in $\mathcal{N}_e(S) = \mathit{RelProd}(\mathcal{R}_e, S)$, where *RelProd* is $\mathit{RelProd}(R,S) = \{s' | \exists s \in S, (s,s') \in R\}$ [5]. The smaller the partitions, the less computation they need, but there is a minimum imposed by the high-level formalism.

Saturation uses a *special iteration strategy*, which is efficient for asynchronous systems. The construction of the MDD representation of the state space starts by building the MDD representing the initial state ($\mathcal{S}^{init}$). Then the algorithm saturates every node in a bottom-up manner, by applying saturation recursively, as if new states are discovered. Saturation iterates through the MDD nodes and generates the complete state space representation using a node-to-node transitive closure. This way saturation avoids the peak size of the MDD to be much larger than the final size, which is a critical problem in traditional approaches.

Saturation-based Structural Model Checking. Saturation-based structural CTL model checking was first presented in [4], where the authors introduced how the least fixed point operators can be computed with the help of saturation. CTL model checking explores the state space in a backward manner. It constructs the inverse representation $\mathcal{N}^{-1}$ and computes the inverse next state, greatest and least fixed points of the operators. The semantics of the three implemented CTL operators [7] is the following:

- **EX**: $\mathbf{s}^0 \vDash \mathsf{EX}\ p$ iff $\exists \mathbf{s}^1 \in \mathcal{N}(\mathbf{s}^0)$ s.t. $\mathbf{s}^1 \vDash p$. EX corresponds to the function $\mathcal{N}^{-1}$, applying one step backward through the next state relation.
- **EG**: $\mathbf{s}^0 \vDash \mathsf{EG}\ p$ iff $\mathbf{s}^0 \vDash p$ and $\forall n > 0, \exists \mathbf{s}^n \in \mathcal{N}(\mathbf{s}^{n-1})$ s.t. $\mathbf{s}^n \vDash p$ so that there is a strongly connected component containing states satisfying p. This needs a greatest fixed point computation. Saturation cannot be applied directly, however the fixed point computation still benefits from the locality due to decomposition.
- **EU**: $\mathbf{s}^0 \vDash \mathsf{E}[p\ \mathsf{U}\ q]$ iff $\mathbf{s}^0 \vDash p$ and $\exists n > 0, \exists \mathbf{s}^1 \in \mathcal{N}(\mathbf{s}^0), \ldots, \exists \mathbf{s}^n \in \mathcal{N}(\mathbf{s}^{n-1})$ s.t. $\mathbf{s}^n \vDash q$ and $\mathbf{s}^m \vDash p$ for all $m < n$ (or $\mathbf{s}^0 \vDash q$). The states satisfying this property are computed with the following least fixed point: $\mathbf{lfp}\ Z[q \vee (p \wedge \mathsf{EX}\, Z)]$, i.e., we search for a state q reached through only states satisfying p.

3 Bounded and Constrained Saturation

In this section an overview is given of the two saturation-based advanced algorithms integrated in our new approach. *Bounded saturation* is used for state space exploration. *Constrained saturation* is used to restrict structural model checking to the bounded state space. Their integration leads to the saturation-based bounded model checking algorithm, which exploits the efficiency of structural model checking for bounded state spaces.

3.1 Bounded Saturation

It is difficult to exploit the efficiency of saturation for bounded state space exploration, because saturation uses an irregular recursive iteration order, which is completely different from traditional breadth-first traversal. Consequently, bounding the recursive exploration steps of saturation does not necessarily guarantee this bound to be global for the state space representation.

There are different solutions for this problem in the literature, both for globally and locally bounded saturation-based state space generation. We chose one that has already proved its efficiency [12]. Although MDDs provide a highly compact solution for state space representation, bounded saturation needs additional distance information during the traversal. For this reason, our chosen algorithm uses EDDs instead of MDDs, and —in addition to the state space— it also encodes the minimal distance of each state from the initial state(s). The algorithm first iterates through the state space until a given bound is reached. After that it cuts the parts that are beyond the depth of the traversal from the EDD, thereby computing the reachability set below the bound. The algorithms in [12] use multiple different cutting strategies. In this work we use the strategy enforcing strict bounds (*TruncateExact* method) extended with our caching mechanism from [11].

3.2 Constrained Saturation

In [13] the authors introduced an advanced saturation-based iteration strategy for the purpose of structural model checking. The algorithm, called *constrained saturation*, computes the least fixed point of the reachability relation that satisfies a given constraint.

The main novelty of the new algorithm is the slightly different iteration style. Instead of combining saturation with breadth-first traversal, it uses a *pre-checking* phase. The algorithm builds on the following observation: in order to do the symbolic step $\mathcal{N}_e$ from the set of states S to a set of states satisfying the constraint C (represented by an MDD), we have to compute $\mathcal{N}_e(S) \cap C$. This contains an expensive intersection operation after each step. Using the following observation: $\mathcal{N}_e(S) \cap C = \mathit{RelProd}(\mathcal{R}_e, S) \cap C = \mathit{RelProd}(\{(\mathbf{s}, \mathbf{s}') | \mathcal{R}_e(\mathbf{s}, \mathbf{s}') \wedge \mathbf{s}' \in C\}, S)$, the algorithm can use pre-checking phase and avoid the computation-intensive intersection operation after the symbolic state space step [13], simply skipping the steps that "go out" of the constraint. Researches showed that the constrained saturation is faster than traditional saturation when there is a constraint on the possible states. This is the situation e.g. in the case of the EU CTL operator.

4 Efficient Methods for Saturation-based Bounded Model Checking

In this section, a new, saturation-based bounded model checking algorithm is presented. The classical saturation-based, non-bounded model checking consists of

two consecutive steps: *1)* state space exploration; and *2)* evaluation of the CTL formula on the explored state space.

The bounded saturation-based model checking (see Fig. 2) performs these steps in an iterative way, where the state space exploration is done until a given bound b that is incremented in each iteration [11]. The input of this iterative algorithm is the initial bound B and an increment value δ: after each iteration, model checking analyses the property on the bounded state space and from the result it makes a *termination decision* if the iteration has to be continued. If the search is continued, then the procedure is restarted with an incremented depth $b := b + \delta$.

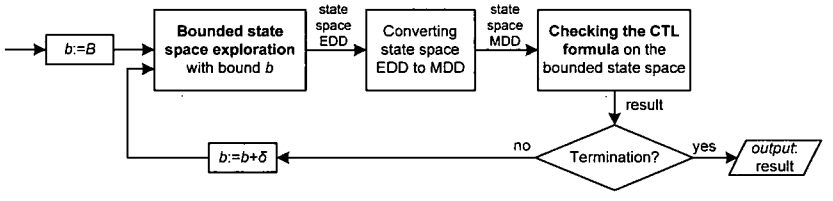


Figure 2: Overview of the saturation-based bounded model checking

Section 4.1 provides various methods for the *bounded state space exploration* using saturation. Section 4.2 describes a new, hybrid state space exploration approach combining EDDs and MDDs. Section 4.3 improves the CTL *formula checking* phase. Section 4.4 is dedicated to the discussion of the termination *decision mechanism*. Finally, Section 4.5 summarizes the contributions.

4.1 State Space Exploration Strategies

The used search strategy for the bounded model checking has a significant impact on performance. In this section different possible search strategy alternatives are introduced and compared. With regard to bounded state space generation, there are two main approaches: *(1)* Ideally, a fixed bound b can be given. The b-bounded state space is explored and the specification is evaluated on it. We call it the *fixed bound strategy*. This is typically not possible, as the necessary b is not known a priori. *(2)* Given an initial bound B and increment value δ, the state space is explored incrementally with bound $b = B + (n-1) \cdot \delta$ in each iteration n. The procedure stops when the model checking question is answered, or it runs out of resources. We call it the *incremental strategy*.

Traditionally, bounded model checking uses the incremental strategy, typically looking one step further ($\delta = 1$) in the state space in a breadth-first manner. According to the characteristics of the saturation iteration strategy, our experiences show that it is better to let saturation increase the depth by at least 5–10 steps. Finding a good trade-off in choosing the iteration depth is important. A one-step iteration would lose the efficiency of saturation. On the other hand, a too large increase of search depth results in the loss of the efficiency of the bounded iteration strategy.

We have developed two basic incremental search strategies:

- The *restarting strategy* starts again from the initial state after each iteration, and uses the increased bound in the exploration [11].
- The *continuing strategy* reuses the formerly explored bounded state space as the set of initial states in the next iteration, and extends it using the bounded saturation algorithm to represent the state space of the increased bound.

The restarting strategy is straightforward to implement and builds mainly on the efficiency of saturation. It simply uses the bounded saturation algorithm with growing bound values. We refer the reader to our previous work [11]. For the continuing strategy we had to modify the bottom-up building mechanism of the saturation algorithm. For this purpose, we needed to extend the algorithm to be able to handle even huge initial state sets. This extension contains the modification of the truncating operations, the caching mechanisms in order to preserve correctness, and the construction of the decision diagram representation to be able to handle huge initial set of states. The continuing strategy can reuse more from the formerly built data structures, making the algorithm typically more efficient than the restarting algorithm.

The result of the bounded state space exploration step is an EDD containing all the states that are within the current bound b, and the next-state relation $\mathcal{R}$ encoded by an MDD. These artefacts are exactly the same for both incremental strategies.

The next section describes a third, hybrid incremental search strategy, the compacting strategy.

4.2 Compacting Strategy

The two incremental state space exploration strategies introduced in the previous section have a common weakness: they store the whole $[0; b]$ part of the state space (i.e., the states with distance between 0 and b; marked as $\mathcal{S}_{[0;b]}$ in the following) in a single EDD. The algorithms use EDD for storing the distance information from the initial state for each state, which is not possible in case of the MDD encoding. This information is necessary to limit the state space exploration at the bound b. The storage of the distance has its price to pay: the EDD representation of the state space is typically less compact than the MDD representation (compare for example Fig. 1(b) and Fig. 1(c)).

While it is unavoidable to store distance information for some states to enforce the bound b, it is not necessary for all the states. Consider the nth iteration ($n > 1$). In this case, the $\mathcal{S}_{[0;b']}$ should be the result of the bounded state space exploration (where $b' = B + (n-1) \cdot \delta$). However, the $\mathcal{S}_{[0;b'-\delta]}$ part was already explored in the previous iteration, and it is known that these states will be present in $\mathcal{S}_{[0;b']}$ too. These states can be stored in the more compact MDD format. The main idea of the new so-called *compacting saturation* method is to store the state spaces discovered in previous iterations using a more compact, MDD-based representation.

Initial State Set. In the previous exploration strategies, each iteration was started either from the initial state (fixed bound strategy, restarting strategy), or from the state space of the previous iteration (that is $\mathcal{S}_{[0;b'-\delta]}$). The goal of compacting saturation is to avoid the EDD encoding for states $\mathcal{S}_{[0;b'-\delta]}$, if possible. Therefore this exploration cannot be started from the initial state of from $\mathcal{S}_{[0;b'-\delta]}$, as it would require the EDD encoding of these states.

It is easy to see that starting the algorithm from $\mathcal{S}_{[0;b'-\delta]}$ is unnecessary. Obviously, $\mathcal{N}(\mathcal{S}_{[0;b'-\delta-1]}) \subseteq \mathcal{S}_{[0;b'-\delta]}$. Therefore new states can only be found from the states on the "border of the bounded state space", i.e., from the states in $\mathcal{S}_{[b'-\delta;b'-\delta]}$, this will be the initial state set of the next iteration.

Details of the Compacting Saturation. In the following, the steps of the new compacting saturation algorithm are described in details.

The *first iteration* of the compacting saturation is started from the state (set) $\mathcal{I}_1 = \mathcal{S}^{init}$ and it explores the state space until the bound $b = B$. The result is the state set $\mathcal{S}_{[0;b]}$ encoded by EDD. Then the algorithm computes the border of the state space, i.e., $\mathcal{S}_{[b;b]} = \mathcal{C}_1$ and converts the EDD of $\mathcal{S}_{[0;b]}$ to an MDD representation $\mathcal{M} = \mathcal{M}_1$. The CTL formula will be evaluated on $\mathcal{M}$.

After, in *each iteration* $n > 1$ where the bound is $b' = B + (n-1) \cdot \delta$, the exploration is started from $\mathcal{I}_n = \mathcal{C}_{n-1} = \mathcal{S}_{[b'-\delta;b'-\delta]}$. The state space is explored until the bound b' that is $\mathcal{S}_{[b'-\delta;b']}$. Next, the algorithm computes the border of the state space $\mathcal{C}_n = \mathcal{S}_{[b';b']}$ and converts the EDD of the state space explored in the current iteration ($\mathcal{S}_{[b'-\delta;b']}$) to an MDD representation $\mathcal{M}_n$. The MDD $\mathcal{M}$ is modified in order to include $\mathcal{M}_n$ too, to represent $\mathcal{M} = \bigcup_{i=1}^{n} \mathcal{M}_i$. The evaluation of the CTL formula will be performed on this $\mathcal{M}$.

The state sets $\mathcal{C}_i$ can be computed using a modified version of the truncation operators described in [12].

Avoiding to Revisit States. During the traversal the previously explored states ($\mathcal{M}$) should not be re-explored: no states of $\bigcup_{i=1}^{n} \mathcal{M}_i$ should be explored again in iteration $n+1$. To be more precise, the goal of the algorithms is the following to keep: $\mathcal{M}_{n+1} \cap (\bigcup_{i=1}^{n} \mathcal{M}_i \setminus \mathcal{I}_{n+1}) = \emptyset$ (overlap in the initial state set is allowed). This is an obvious consequence of the state space parts defined above, i.e., $\mathcal{S}_{[b;b+\delta]} \cap \mathcal{S}_{[b+\delta;b+2\delta]} = \mathcal{S}_{[b+\delta;b+\delta]}$. However, if a state $\mathbf{s} \in \mathcal{M}_j$ $(j < n)$ is reachable from $\mathcal{I}_n$, it can be explored during the iteration n causing that it will be part of $\mathcal{M}_n$ too. It effectively means that the state $\mathbf{s}$ will be stored with two different distance value. The algorithm has to prevent this situation for efficiency reasons and also to keep the correctness.

A solution could be to subtract $\mathcal{M}$ from the state set of the current iteration after each step. It would make the method correct, but not efficient. The problem is similar to the motivation of the constrained saturation, except that here the forbidden states should be excluded, not the set of allowed states should be respected. Using the observation of constrained saturation, the following will be true: $\mathcal{N}_e(S) \setminus \mathcal{X} = \mathit{RelProd}(\mathcal{R}_e, S) \setminus \mathcal{X} = \mathit{RelProd}(\{(\mathbf{s}, \mathbf{s}') | \mathcal{R}_e(\mathbf{s}, \mathbf{s}') \wedge \mathbf{s}' \notin \mathcal{X}\}, S)$

If we use the observation above with $\mathcal{X} = \mathcal{M}$ as set of forbidden states, it can be efficiently avoided to reexplore states that were already explored in previous iterations.

4.3 Constrained Saturation using the Bounded State Space

Many model checking tools limit the syntax of the specification to a subset of the CTL temporal language, in order to simplify the analysis task and to boost the performance. We want to support the full semantics of CTL in model checking, and thus we must use backward traversal. This is our main reason for choosing the traditional, fixed point-based algorithms; as the semantics of forward and backward CTL model checking are different (and incomparable) [8].

The naive approach to combine bounded exploration and structural model checking would be to apply the fixed point computations from the bounded state space on the complete lattice. However, the efficiency of this approach would converge to traditional fixed point computations. Due to the symbolic representation of the saturation algorithm, it would be possible to check certain states that are not inside the explored, bounded part of the state space. It could be improved by constructing the intersection of the result of fixed point iterations with the bounded state space representation, but this still suffers from poor performance due to the extensive use of the costly intersection operation.

Our aim is to exploit the constrained saturation iteration strategy to provide an efficient bounded model checking algorithm. The symbolically encoded explored bounded state space can serve as the constraint in the constrained saturation algorithm. This way we can expeditiously bound the least fixed point computations: instead of employing the intersection operation after each step (image computation), the algorithms apply intersection only at the beginning of the fixed point computations (when they compute the inputs of the constrained saturation algorithm). Below we define how the constrained saturation decides on the essential CTL operators (where **lfp** denotes the least fixed point, and *bss* denotes the bounded state space as stored by the MDD):

- **EF**: $M, s \vDash_k \mathsf{EF}\, p$ iff $s_0 \subseteq \mathbf{lfp}\ Z[(p \wedge bss) \vee (bss \wedge \mathsf{EX}\, Z)] = ConsSaturation(bss, p \cap bss)$[1]. This way we can directly exploit the constrained saturation algorithm to produce the least fixed point in the given bounded state space *bss*. The fixed point computational traversal explores only states inside *bss*, while the only intersection operation is used at the computation of the input argument: $p \cap bss$. The result can be utilised by other, both least and greatest fixed point operators.

- **EU**: $M, s \vDash_k \mathsf{E}[p\, \mathsf{U}\, q]$ iff $s_0 \subseteq \mathbf{lfp}\ Z[(q \wedge bss) \vee (bss \wedge p \wedge \mathsf{EX}\, Z)] = ConsSaturation(bss \cap q, bss \cap p)$. This is similar to using the constrained saturation algorithm in traditional saturation-based model checking [13], but within a bounded setting. The fixed point computational traversal explores only states

[1] For the pseudocode of the *ConsSaturation* function we refer the reader to [13].

inside *bss*, while the only intersection operations are used at the computation of the input arguments: $bss \cap p$ and $bss \cap q$. This result can also be nested into both least and greatest fixed point operators.

As greatest fixed point computations (EG) and simple next state operators (EX) do not require such restrictions in the exploration, we apply traditional fixed point algorithms for them. Although operator EF is just a special case of operator EU, for performance reasons it is worth to be implemented separately.

4.4 Decision Mechanism

It is also necessary to be able to decide if an answer can be given to the requirement checked. The bounded model checking is a semi-decision procedure, therefore it can be used to ensure the following behavioural properties of the specification:

- *Invariant or safety*: proving these properties needs either the full state space to be explored, or bounded model checking can give a finite counterexample (or witness), if it exists.
- *Liveness*: bounded model checking can either find a *lasso-shaped* trace as counterexample (or witness) to these properties, or the full state space has to be explored to evaluate them.
- Other properties, like combination of safety and liveness properties: 3-valued logic can be used for decision. We refer the reader to our previous work [11].

Obviously, the bounded model checking can also be terminated if the full state space is already explored. In the case of the compacting algorithm, this can be detected easily by checking if $\mathcal{C}_n = \emptyset$. If $\mathcal{C}_n = \emptyset$, the algorithm terminates.

Invariant and safety properties are usually proved (by symbolic model checking) by finding inductive invariants without exploring the full state space. This approach cannot be used directly for liveness properties.

Finding Inductive Proof Against Liveness Properties. The EDD-based state space representation helps us to tell more about liveness properties. Refuting liveness properties may come from the fact that: either (1) the algorithm has to explore more from the state space to find a witness, or (2) the liveness property does not hold, and there exists a counterexample in the bounded state space.

Our approach can handle these differences. This is in contrast to non-saturation-based bounded model checking approaches, since they have to encode the difference of the two cases into the SAT formula directly, which is inefficient. If a liveness property $\mathsf{EG}\, p$ does not hold in the bounded state space *bss*, we can decide whether to investigate the state space further, or to conclude that it will never hold. Let $p_{d=bound}$ be the set of states, where p is true and their distance from the initial state is $d = bound$. $p_{d=bound}$ is encoded in the EDD, we need to traverse it only once to get this state set. It can be computed efficiently from the symbolic encoding. Let $result = \mathbf{lfp}\ Z[p_{d=bound} \vee (p \wedge \mathsf{EX}\, Z)] = ConsSaturate(p, p_{d=bound})$, thus *result* is

the set of states from which the $p_{d=bound}$ states can be reached only through states where p holds. If the initial state is not in the set *result*, the evaluation of $\mathsf{EG}\,p$ can be finished, as the result cannot be true: $s_0 \notin result \Rightarrow s_0 \not\models \mathsf{EG}\,p$.

4.5 Summary of the Contributions

In this section we described improvements of the saturation-based bounded model checking algorithm. New incremental state space exploration algorithms were presented (continuing and compacting saturation) that can reduce the overhead of the incremental methods compared to the fixed bound strategy.

With the novel use of constrained saturation, the algorithm avoids to examine states outside of the discovered bounded state space in the model checking phase without the need of expensive intersection operators. This way $\forall f(Z)$: $\mathbf{fp}\ f(Z) \subseteq bss$, for all fixed point the bounded saturation algorithm is bounded by the state space, even for the least fixed point computations.

5 Evaluation

We have performed many measurements in order to examine the efficiency of our new algorithm and compare it to a classical saturation-driven structural model checking algorithm. We have also examined the scalability of the new approach and compared to the former one. For this purpose we have developed an experimental implementation using the C# programming language of the fixed bound strategy (*B-Fix*), the restarting incremental strategy (*B-I-Rest*), the continuing incremental strategy (*B-I-Cont*), and for the compacting saturation (*B-I-Comp*). We have also implemented the algorithm taken from [13] as the reference for comparison, which we denoted in the measurements as "*Unlim*". For the measurements we used a desktop PC (Intel Core i7-3770 3.4 GHz CPU, 8 GB memory with Windows 7 x64 and .NET 4.0 framework).

5.1 Measurements on Benchmark Models

This section shows measurements on benchmark models widely known in the model checking community. We used the models[2] of Dining Philosophers (DPhil–N, [5]), Flexible Manufacturing System (FMS–N, [5]), Tower of Hanoi (Hanoi–N, [11]), an assembly line using Kanban method (Kanban–N, [5]), the Round Robin protocol (RR–N, [3]), and the Slotted Ring protocol (SR–N, [3]).

Both the initial bound B and the increment distance δ are input parameters, thus our algorithm can be fine-tuned by the user. If the properties to be proven are expected to be "shallow", then the algorithm can be set to work optimally for smaller distances. On the other hand, when the properties to prove are "deeper", then both the initial bound and the increment distance can be set bigger to find a

[2]The models can be downloaded from `http://petridotnet.inf.mit.bme.hu/publications/AC2014-Saturation_DarvasEtAl_models.zip`

proof in fewer iterations. A priori knowledge about the expected behaviour of the properties can significantly reduce the computational time.

Table 1 lists the runtime measurements for various CTL expressions on different models with different parameters.

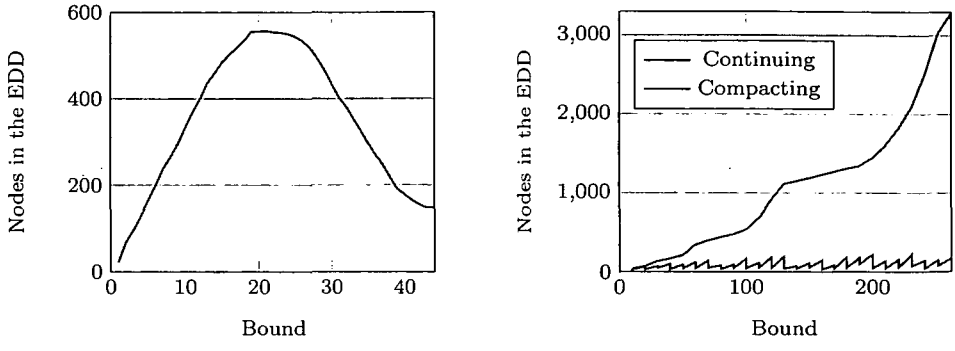
The Dining Philosophers (DPhil-N, where N is the number of philosophers) model revealed that for those models, where the saturation algorithm answers the model checking question quickly, the overhead of bounded model checking does not pay off. This is also the case for the chosen requirements for the Round Robin (RR-N) models, where the traditional saturation algorithm is highly efficient. These models are built from many identical components with relatively small local state spaces: the saturation algorithms traverses the possible state spaces very fast.

As can be seen in Table 1, saturation-based model checking can be highly efficient for asynchronous systems, and the modified bounded iteration strategy requires more computational resources, so one would expect that for these (especially for models DPhil and RR and SR) models the traditional approach performs better. However, the SR-N models (where N is the number of communication nodes) shows the advantage of bounded model checking, as traditional model checking has high runtimes even for a relatively simple property. This is a typical use case for bounded model checking as the property could be verified in a relatively small depth. The SR-N models have quite complex behaviour, where bounded model checking can significantly decrease the verification time.

We have measured the runtime for also synchronous models as the efficiency of the algorithms depends also on the extent of the synchronization in the systems. The verification goal was a combined safety-liveness property ($\mathsf{EG}(\mathsf{EF}(B_{\downarrow 8} = 1))$ of the Towers of Hanoi models, where $B_{\downarrow 8} = 1$ denotes the placement of the 8th disk to the 2nd rod, see the Petri net models for further details). The traditional structural model checking approach (Unbounded) runs out of resources early. Running the algorithm parameterised by the formerly computed bound (being enough to answer the verification question) (Bounded, fixed bound) has the best runtime results. The continuing strategy has advantage over the restarting strategy, as it uses up the formerly computed results during the model checking.

Similar results can be observed for the FMS model. The measurements of the FMS models also used a combined safety-liveness property that represents the existence of a cycle in a certain set of states satisfying safety requirements (based on [4]). The structural model checking algorithm time-outs for large parameters. For small values of N the unbounded method had the lowest runtime, but for increasing parameters the compacting saturation outperformed both the unlimited and the other bounded algorithms. The compacting method provided better results than the B-Fix method thanks to the efficient state space representation.

In some cases, like in the case of the FMS or the Kanban model with the chosen requirements, the restarting strategy solves the model checking problem for every parameter *faster* than the continuing strategy. We investigated the reason: as it is depicted in Figure 3(a), the state space representation of asynchronous systems (like FMS) grows steeply up to a maximal value (557 nodes), but after that it starts decreasing (resembling a bell curve) until 148 that is the final size of the state space



(a) FMS-3 (b) Hanoi-8

Figure 3: Size of state space representation (EDD) at each iteration

Table 1: Run times of CTL expression evaluation

	Run time[3] [s]				
N	**Unlim**	**B-I-Rest**	**B-I-Cont**	**B-I-Comp**	**B-Fix**
(1) DPhil–N, expression: $\mathsf{E}(\neg \mathit{eating}_2 \,\mathsf{U}\, \mathit{eating}_1)$, $B = 20$, $\delta = 10$)					
10	**0.09**	0.10	0.09	0.33	*0.04*
50	**0.12**	5.02	5.12	1.81	*0.58*
100	**0.22**	12.78	12.92	3.88	*1.45*
(2) FMS–N, expression: $\mathsf{EG}(\mathsf{E}(M1 > 0 \,\mathsf{U}\, (P1s = P2s = P3s = 3)))$, $B = 20$, $\delta = 10$)					
25	**1.00**	40.02	40.57	18.65	*37.39*
50	**5.54**	56.31	59.51	21.96	*44.99*
100	48.33	61.68	63.51	**21.78**	*47.03*
1000	>600	60.17	62.12	**21.69**	*47.67*
10,000	>600	60.06	62.40	**21.59**	*54.34*
(3) Hanoi–N, expression: $\mathsf{EG}(\mathsf{EF}(B_{\downarrow 8} = 1))$, $B = 20$, $\delta = 10$)					
12	21.97	2.49	**0.81**	1.55	*0.36*
14	>600	2.89	**0.98**	2.39	*0.46*
16	>600	3.25	**1.11**	1.83	*0.49*
18	>600	3.60	**1.22**	1.91	*0.56*
20	>600	4.08	**1.43**	2.14	*0.67*
(4) Kanban–N, expression: $\mathsf{EF}(\mathit{pout}_4 = 5)$, $B = 20$, $\delta = 10$)					
30	**0.31**	1.01	1.29	3.11	*0.53*
50	1.91	**1.36**	1.73	3.65	*0.69*
100	26.51	**1.37**	1.78	3.64	*0.67*
200	>600	**1.37**	1.74	3.68	*0.68*
(5) RoundRobin–N (RR–N), expression: $\mathsf{EG}(\mathit{true})$, $B = 10$, $\delta = 5$)					
10	**0.04**	0.21	0.21	0.71	*0.05*
25	**0.31**	3.65	2.40	23.72	*1.29*
50	**2.05**	68.26	32.40	>300	*12.49*
(6) SlottedRing–N (SR–N), expression: $\mathsf{EF}(E_2 = 1 \wedge A_2 = 1)$, $B = 10$, $\delta = 5$)					
100	7.59	**0.89**	0.92	2.95	*0.84*
200	60.08	**2.31**	2.37	8.94	*2.29*
300	294.76	**4.31**	4.43	18.34	*4.30*

EDD. The continuing strategy uses these intermediate state space representations as the initial state, which can be a large computational overhead compared to starting the iteration from the initial state. By beginning model checking from scratch (i.e., using the restarting strategy) we can exploit the efficiency of saturation for building the state space representation. By modifying an intermediate representation (i.e., using the continuing strategy) the algorithm has to do more computation, especially if the intermediate representation is larger than the final one.

The Kanban model shows the superiority of the bounded approaches compared to the full state space exploration. In addition, we must emphasize here that the new model checking algorithm is also superior to the former one of [11]. Constrained saturation-based model checking could reduce the runtime of the verification of properties from [11] on model Kanban–30 by approximately 70%. The reachability property in Table 1 required 90% less time to be verified by the new CTL model checking algorithm showing that we could efficiently utilize constrained saturation in bounded model checking.

We also analysed the EDD sizes of the Hanoi model (see Figure 3(b)) (representing synchronous models). In this case, the size of the state space representation grows constantly using the continuing strategy. However, the compacting saturation can effectively reduce the size of the EDD representation by storing the states discovered in previous iterations using MDD. That causes this sawtooth pattern.

5.2 Measurements on an Industrial Case Study

To evaluate our contribution we performed measurements on a model describing a real, industrial safety function. Our case study is a safety function included within the Reactor Protection System of a nuclear power plant [10][4]. This safety function initiates an emergency operation in case of a predefined chain of events happens. The detection of the specific event chain requires a complex logic, the design of which is error prone. This also puts emphasis on the necessity of using formal verification to ensure correctness.

The safety function receives inputs from different sensors, and computes the values of outputs, one of which initiates the emergency protection action. The values of the outputs depend on the recent and past values of the inputs, and some internal timers. The design of the controller was specified by simple combinatorial (OR gates, AND gates, and inverters), and sequential (SR flip-flops, delay and pulse modules) function blocks. The proper combination of these logic elements is required to guarantee that the emergency protection action will be initiated only in case of a specific dangerous event happened.

We have created a hierarchical coloured Petri net model of this safety logic. The structure of the CPN model preserves the data flow characteristics of the function block description. The subnets of the CPN model are the models for the functional modules. After the subnets have been derived and verified separately, they only had to be connected together properly [1]. The model can be parameterized, thus here

[3]The abbreviations used in the table headers are described in the first paragraph of Section 5.

[4]The authors of [10] decomposed the system and have done manual compositional verification.

Table 2: Run times of CTL expression evaluation on PRISE models

Model	Unlim	B-I-Rest	B-I-Cont	B-I-Comp
Expression 1: *OUTPUT-1 can be true.*				
PRISE S	2.84 s	**1.14** s	1.21 s	1.34 s
PRISE M	11.18 s	13.66 s	8.91 s	**4.79** s
PRISE L	27.88 s	13.72 s	8.98 s	**4.80** s
Expression 2: *There is no emergency action, if not necessary.*				
PRISE S	**5.37** s	33.97	19.98 s	12.18 s
PRISE M	**21.96** s	> 1800 s	113.03 s	30.38 s
PRISE L	56.86 s	> 1800 s	433.97 s	**51.83** s

we measured three different models with different complexity (denoted by PRISE S, M, and L).

The measurements being presented in Table 2 show that if the verification requirement is "shallow" (Expression 1), then the bounded algorithms provide significantly lower runtime than the unbounded algorithm, especially the compacting method. Expression 2 requires the exploration of a relatively big part of the state space. Therefore the bounded algorithms are slower than the unbounded algorithm. However, the compacting scales better and it provides slightly better runtime than the unbounded method for PRISE L. Also, the continuing strategy provides better results than the restarting strategy. Furthermore the restarting strategy runs out of memory for the PRISE M and L model, while the continuing and compacting strategy finish the execution for both models.

6 Conclusion and Future Work

In this paper an advanced bounded model checking approach based on the saturation algorithm was presented. It exploits the efficiency of saturation and enables us to verify complex, or in certain cases even infinite-state models. This approach also extends the set of asynchronous systems that can be analysed with symbolic methods. The efficiency of the new approach was proved by measurements.

We intend to develop the presented solutions further. We will investigate the use of forward model checking instead of the classical backward fixed point computation, as we believe this can further improve the performance of our algorithm. We also plan to use the constrained saturation algorithm in a different way, in order to avoid redundant computations more efficiently.

References

[1] Bartha, T., Vörös, A., Jámbor, A., and Darvas, D. Verification of an industrial safety function using coloured Petri nets and model checking. In *Proc. of the*

14th Int. Conf. on Modern Information Technology in the Innovation Processes of the Industrial Enterprises, pages 472–485. MTA SZTAKI, 2012.

[2] Biere, A., Cimatti, A., Clarke, E.M., and Zhu, Y. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer, 1999.

[3] Ciardo, G., Marmorstein, R., and Siminiceanu, R. Saturation unbound. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2619 of *LNCS*, pages 379–393. Springer, 2003.

[4] Ciardo, G. and Siminiceanu, R. Structural symbolic CTL model checking of asynchronous systems. In *Computer Aided Verification*, volume 2725 of *LNCS*, pages 40–53. Springer, 2003.

[5] Ciardo, G., Zhao, Y., and Jin, X. Ten years of saturation: A Petri net perspective. In *Transactions on Petri Nets and Other Models of Concurrency V*, volume 6900 of *LNCS*, pages 51–95. Springer, 2012.

[6] Clarke, E.M., Biere, A., Raimi, R., and Zhu, Y. Bounded model checking using satisfiability solving. *Formal Methods in System Design*, 19(1):7–34, 2001.

[7] Clarke, E.M., Grumberg, O., and Peled, D.A. *Model Checking*. The MIT Press, 1999.

[8] Henzinger, T., Kupferman, O., and Qadeer, S. From pre-historic to post-modern symbolic model checking. In *Computer Aided Verification*, volume 1427 of *LNCS*, pages 195–206. Springer, 1998.

[9] Murata, T. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[10] Németh, E. and Bartha, T. Formal verification of safety functions by reinterpretation of Functional Block based specifications. In *Formal Methods for Industrial Critical Systems*, volume 5596 of *LNCS*. Springer, 2009.

[11] Vörös, A., Darvas, D., and Bartha, T. Bounded saturation-based CTL model checking. *Proceedings of the Estonian Academy of Sciences*, 62(1):59–70, 2013.

[12] Yu, A., Ciardo, G., and Lüttgen, G. Decision-diagram-based techniques for bounded reachability checking of asynchronous systems. *International Journal on Software Tools for Technology Transfer*, 11:117–131, 2009.

[13] Zhao, Y. and Ciardo, G. Symbolic CTL model checking of asynchronous systems using constrained saturation. In *Automated Technology for Verification and Analysis*, volume 5799 of *LNCS*, pages 368–381. Springer, 2009.

Received 10th March 2014

Acta Cybernetica **22** (2016) 591–612.

On Cyber Attacks and the Maximum-Weight Rooted-Subtree Problem

Geir Agnarsson*, Raymond Greenlaw†, and Sanpawat Kantabutra‡

Abstract

This paper makes three contributions to cyber-security research. First, we define a model for cyber-security systems and the concept of a *cyber-security attack* within the model's framework. The model highlights the importance of *game-over components*—critical system components which if acquired will give an adversary the ability to defeat a system completely. The model is based on systems that use defense-in-depth/layered-security approaches, as many systems do. In the model we define the concept of *penetration cost*, which is the cost that must be paid in order to break into the next layer of security. Second, we define natural decision and optimization problems based on cyber-security attacks in terms of doubly weighted trees, and analyze their complexity. More precisely, given a tree T rooted at a vertex r, a *penetrating cost* edge function c on T, a *target-acquisition* vertex function p on T, the attacker's *budget* and the *game-over threshold* $B, G \in \mathbb{Q}^+$ respectively, we consider the problem of determining the existence of a rooted subtree T' of T within the attacker's budget (that is, the sum of the costs of the edges in T' is less than or equal to B) with total acquisition value more than the game-over threshold (that is, the sum of the target values of the nodes in T' is greater than or equal to G). We prove that the general version of this problem is intractable, but does admit a polynomial time approximation scheme. We also analyze the complexity of three restricted versions of the problems, where the penetration cost is the constant function, integer-valued, and rational-valued among a given fixed number of distinct values. Using recursion and dynamic-programming techniques, we show that for constant penetration costs an *optimal* cyber-attack strategy can be found in polynomial time, and for integer-valued and rational-valued penetration costs *optimal* cyber-attack strategies can be found in pseudo-polynomial time. Third, we provide a list of open problems relating to the architectural design of cyber-security systems and to the model.

Keywords: cyber security, defense-in-depth, game over, information se-

*Department of Mathematical Sciences, George Mason University, Fairfax, VA 22030, E-mail: geir@math.gmu.edu

†Cyber Security Studies, United States Naval Academy, Annapolis, Maryland 21402, E-mail: greenlaw@usna.edu

‡Computer Engineering Department, Chiang Mai University, Chiang Mai, 50200, Thailand, E-mail: sanpawat@alumni.tufts.edu

DOI: 10.14232/actacyb.22.3.2016.3

curity, layered security, weighted rooted trees, complexity, polynomial time, pseudo-polynomial time

1 Introduction

Our daily life, economic vitality, and a nation's security depend on a stable, safe, and secure cyberspace. Cyber security is so important that the United States (US) Department of Defense established the US Cyber Command to take charge of pulling together existing cyberspace resources, creating synergy, and synchronizing war-fighting efforts to defend the information-security environment of the US [24]. Other countries also have seen the importance of cyber security. To name just a few in what follows, in response to North Korea's creation of a cyber-warfare unit, South Korea created a cyber-warfare command in December 2009 [23]. During 2010, China introduced its first department dedicated to defensive cyber war and information security in response to the creation of the US Cyber Command [4]. The United Kingdom has also stood up a cyber force [5]. Other countries are quickly following suit.

Cyberspace has become a new frontier that comes with new opportunities, as well as new risks. According to a 2012 study of US companies, the occurrence of cyber attacks has more than doubled over a 3-year period while the adverse financial impact has increased by nearly 40 percent [8]. More specifically, US organizations experienced an average of 50, 72, and 102 successful attacks against them per week in 2010, 2011, and 2012, respectively. In [21] a wide range of cyber-crime statistics are reported, including locations of attacks, motivation behind attacks, and types of attacks. The number of cyber attacks is increasing rapidly, and for the month of June 2013, 4% of attacks were classified as cyber warfare, 8% as cyber espionage, 26% as hacktivism, and 62% as cyber crime (see [21]). Over the past couple of years these percentages have varied significantly from month-to-month. In order to respond to cyber attacks, organizations have spent increasing amounts of time, money, and energy at levels that are now becoming unsustainable. Despite the amounts of time, money, and energy pouring into cyber security, the field is still emerging and widely applicable solutions to the problems in the field have not yet been developed.

A secure system must defend against all possible cyber attacks, including zero-day attacks that have never been known to the defenders. But, due to limited resources, defenders generally develop defense systems for the attacks that they do know about. Their systems are secure to known attacks, but then become insecure as new kinds of attacks emerge, as they do frequently. To build a secure system, therefore, requires first principles of security. "In other words, we need a *science of cyber security* that puts the construction of secure systems onto a firm foundation by giving developers a body of laws for predicting the consequences of design and implementation choices" [19]. To this end Schneider called for more models and abstractions to study cyber security [19]. In his article Schneider suggested building a science of cyber security from existing areas of computer science. In particular,

he mentioned formal methods, fault-tolerance, cryptography, information theory, game theory, and experimental computer science. All of these subfields of computer science are likely to be valuable sources of abstractions and laws.

Cyber security presents many new challenges. Dunlavy et al. discussed what they saw as some of the major mathematical problems in cyber security [9]. One of the main challenges is modeling large-scale networks using explanatory and predictive models. Naturally, graph models were proposed. Some common measures of a graph that such a model would seek to emulate are distribution over the entire graph of vertex in-degrees and out-degrees, graph diameter, community structure, and evolution of any of the mentioned measures over time [6]. Pfleeger discussed a number of useful cyber-security metrics [17]. She introduced an approach to cyber-security measurement that uses a multiple-metrics graph as an organizing structure by depicting the attributes that contribute to overall security, and uses a process query system to test hypotheses about each of the goals based on metrics and underlying models. Rue, Pfleeger, and Ortiz developed a model-evaluation framework that involves making explicit each model's assumptions, required inputs, and applicability conditions [18].

Complexity science, which draws on biological and other natural analogues, seems under utilized, but perhaps is one of the more-promising approaches to understanding problems in the cyber-security domain [3]. Armstrong, Mayo, and Siebenlist suggested that models of complex cyber systems and their emergent behavior are needed to solve the problems arising in cyber security [3]. Additionally, theories and algorithms that use complexity analysis to reduce an attacker's likelihood of success are also needed. Existing work in the fields of fault tolerance and high-reliability systems are applicable too. Shiva, Roy, and Dasgupta proposed a cyber-security model based on game theory [20]. They discovered that their model works well for a dynamically-changing scenario, which often occurs in cyber systems. Those authors considered the interaction between the attacks and the defense mechanisms as a game played between the attacker and the defender.

This paper is our response to the call for more cyber-security models in [19]. This work also draws attention to the importance of designing systems that do not have *game-over components*—components that are so important that once an adversary has taken them over, one's system is doomed. Since, as we will see, such systems can be theoretically hacked fairly efficiently. We model (many known) security systems mathematically and then discuss their vulnerabilities. Our model's focus is on systems having layered security; each security layer possesses valuable assets that are kept in *containers* at different levels. An attacker attempts to break into these layers to obtain assets, paying penetration costs along the way in order to break in, and wins if a given game-over threshold is surpassed before the attacker's budget runs out. A given layer of security might be, for example, a firewall or encryption. The associated cost of by-passing the firewall or encryption is the penetration cost that is used in the model. We formalize the notion of a cyber attack within the framework of the model. For a number of interesting cases we analyze the complexity of developing cyber-attack strategies.

The outline of this article is as follows. In Section 2 we define the model for

cyber-security systems, present an equivalent weighted-tree view of the model, and define natural problems related to the model. A general decision problem (Game-Over Attack Strategy, Decision Problem GOAS-DP) based on the model is proved NP-complete in Section 3; its corresponding optimization problem (GOAS-OP) is NP-hard. In sections 4, 5, and 6 we provide a polynomial-time algorithm for solving GOAS-OP when penetration costs are constant, a pseudo-polynomial-time algorithm for solving GOAS-OP when penetration costs are integers, a polynomial-time approximation algorithm for solving GOAS-OP in general, and a polynomial-time algorithm for solving GOAS-OP when penetration costs are rational numbers from a prescribed finite collection of possible rational costs, respectively. As an easy corollary, we obtain a pseudo-polynomial-time algorithm for solving an optimization problem on general weighted non-rooted trees. Table 1 summarizes the computational results of the paper. Conclusions and open problems are discussed in Section 7.

Problem Name	Time	Class
GOAS-DP	–	NP-complete
GOAS-OP	–	NP-hard
GOAS-DP constant pc	$O(m^2n)$	P
GOAS-OP constant pc	$O(m^2n)$	P
GOAS-DP integer pc	$O(B^2n)$	pseudo-pt
GOAS-OP integer pc	$O(B^2n)$	pseudo-pt
GOAS-OP approx.	$O((1/\epsilon)^2n^3)$	P
GOAS-DP rational pc	$O(m^{2d}n)$	P
GOAS-OP rational pc	$O(m^{2d}n)$	P

Table 1: Summary of results about the cyber-security model contained in the paper. Note that in the table "pc" stands for "penetration cost," and "pseudo-pt" stands for pseudo-polynomial time. The values of m, n, B, and d are as given in the respective theorems.

2 Model for Cyber-Security Systems

2.1 Basic Setup

When defining our cyber-security game-over model, we need to strike a balance between simplicity and utility. If the model is too simple, it will not be useful to provide insight into real situations; if the model is too complex, it will be cumbersome to apply, and we may get bogged down in too many details to see the forest from the trees. In consultation with numerous cyber-security experts, computer scientists, and others, we have come up with a good compromise for our model between ease-of-use and the capability of providing useful insights.

Many systems contain layered security or what is commonly referred to as

defense-in-depth, where valuable assets are hidden behind many different layers or secured in numerous ways. For example, a *host-based defense* might layer security by using tools such as signature-based vendor anti-virus software, host-based systems security, host-based intrusion-prevention systems, host-based firewalls, encryption, and restriction policies, whereas a *network-based defense* might provide defense-in-depth by using items such as web proxies, intrusion-prevention systems, firewalls, router-access control lists, encryption, and filters [14]. To break into such a system and steal a valuable asset requires several levels of security to be penetrated. Our model focuses on this layered aspect of security and is intended to capture the notion that there is a cost associated with penetrating each additional level of a system and that attackers have finite resources to utilize in a cyber attack. We also build the concept of critical game-over components.

2.2 Definition of the Cyber-Security Game-Over Model

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$, $\mathbb{Q}$ be the rational numbers, and $\mathbb{Q}^+$ be the positive rational numbers. With the intuition provided in the previous section in mind, we now present the formal definition of the model.

Definition 2.1. *A* cyber-security game-over model M *is a six-tuple* $(\mathcal{T}, \mathcal{C}, \mathcal{D}, \mathcal{L}, B, G)$, *where*

1. *The set* $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ *is a collection of* targets, *where* $k \in \mathbb{N}$. *The value* k *is the* number of targets. *Corresponding to each target* t_i, *for* $1 \leq i \leq k$, *is an associated* target acquisition value $v(t_i)$, *where* $v(t_i) \in \mathbb{Q}$. *We also refer to the target acquisition value as the* acquisition value *for short, or as the* reward *or* prize.

2. *The set* $\mathcal{C} = \{c_1, c_2, \ldots, c_l\}$ *is a collection of* containers, *where* $l \in \mathbb{N}$. *The value* l *is the* number of containers. *Corresponding to each container* c_i, *for* $1 \leq i \leq l$, *is an associated* penetration cost $p(c_i)$, *where* $p(c_i) \in \mathbb{Q}$.

3. *The set* $\mathcal{D} = \{C_1, C_2, \ldots, C_l\}$ *is the set of* container nestings. *The tuple* C_i, *for* $1 \leq i \leq l$, *is called the* penetration list *for container* c_i *and is a list in left-to-right order of containers that must be penetrated before* c_i *can be penetrated. If a container* c_i *has an empty penetration list, and its cost* $p(c_i)$ has been paid, *we say that the* container has been penetrated. *If a container* c_i *has a non-empty penetration list and each container in its list has been* penetrated *in left-to-right order, and its cost* $p(c_i)$ *has been paid, we say that the* container has been penetrated. *The number of items in the tuple* C_i *is referred to as the* depth of penetration required for C_i. *If container* c_j *appears in* c_i's *tuple* C_i, *we say that container* c_i is dependent on container c_j. *If there are no two containers* c_i *and* c_j *such that container* c_i *is dependent on container* c_j *and container* c_j *is dependent on container* c_i, *then we say the* model is well-formed.

4. *The set* $\mathcal{L} = \{l_1, l_2, \ldots, l_k\}$ *is a list of container names. These containers specify the* level-1 locations *of the targets. For* $1 \leq i \leq k$ *if target* t_i *has level-1 location* l_i*, this means that there is no other container* $\widehat{c}$ *such that container* $\widehat{c}$ *is dependent on container* l_i *and container* $\widehat{c}$ *contains target* t_i*. Target* t_i *is said to be* located at level-1 in container l_i*. The target* t_i *is also said to be* located in container l_i *or any container on which container* l_i *is dependent. When a target's level-1 container has been penetrated, we say that the* target has been acquired.

5. *The value* $B \in \mathbb{Q}$ *is the* attacker's budget. *The value represents the amount of resources that an attacker can spend on a cyber attack.*

6. *The value* $G \in \mathbb{Q}$ *is the* game-over threshold *signifying when critical components have been acquired.*

The focus of this paper is on cyber-security game-over models that are well-formed, which are motivated by real-world scenarios. In the next section we introduce a graph-theoretic version of the model using weighted trees.

Remark. (i) In part 3 of the definition we refer to the cost of a container c_i being paid. By this we simply mean that $p(c_i)$ has been deducted from the remaining budget, B', and we require that $B' - p(c_i) \geq 0$. (ii) In part 4 of the definition we maintain a general notion of containment for targets by specifying the inner-most container in which a target is located. Although containers can have partial overlap, we require that the inner-most container be unique.

In the next definition we formalize the notion of a *cyber-security attack strategy.*

Definition 2.2. *A* cyber-security attack strategy *in a cyber-security game-over model* M *is a list of containers* $c_1, c_2, \ldots, c_r$ *from* M*. The* cost of an attack strategy *is* $\sum_{i=1}^{r} p(c_i)$*. A* valid attack strategy *is one in which the penetration order is not violated. A* game-over attack strategy in a cyber-security game-over model M *is a valid attack strategy* $c_1, c_2, \ldots, c_r$ *whose cost is less than or equal to* B *and whose total target acquisition value* $\sum_{i=1}^{r} v(t_i) \geq G$*. We call such a game-over attack strategy in a cyber-security game-over model a* (successful) cyber-security attack *or* cyber attack *for short.*

Note that this notion of a cyber attack is more general than some, and, for example, espionage would qualify as a cyber attack under this definition. The definition does not require that a service or network be destroyed or disrupted. Since many researchers will think of Definition 2.1 from a graph-theory point of view, in the next section we offer that perspective. As we will soon see, the graph-theoretic perspective allows us to work more easily with the model mathematically and to relate to other known results.

2.3 Game-Over Model in Terms of Weighted Trees

In this section we describe the (well-formed) game-over model in terms of weighted trees. The set $\mathcal{D}$ of nested containers in Definition 2.1 has a natural rooted-tree

structure, where each container corresponds to a vertex that is not the root, and we have an edge from a parent u down to a child v if and only if the corresponding container $c(u)$ includes the container $c(v)$ in it. The weight of an edge from a parent to a child represents the cost of penetrating the corresponding container. The weight of a vertex represents the acquisition value/prize/reward obtained by penetrating/breaking into that container.

Sometimes we don't distinguish a target from its acquisition value/prize/reward nor a container from its penetration cost. We can assume that the number of containers and targets is the same. Since if we have a container housing another container (and nothing else), we can just look at this "double" container as a single container of penetration cost equal to the sum of the two nested ones. Also, if a container contains many prizes, we can just lump them all into a single prize, which is the sum of them all. The following is a graph-theoretic version of Definition 2.1.

Definition 2.3. *A* cyber-security (game-over) model (CSM) M *is given by an ordered five tuple* $M = (T, c, p, B, G)$, *where* T *is a tree rooted at* r *having* $n \in \mathbb{N}$ *non-root vertices,* $c : E(T) \to \mathbb{Q}$ *is a penetration-cost weight function,* $p : V(T) \to \mathbb{Q}$ *is the target-acquisition-value weight function, and* $B, G \in \mathbb{Q}^+$ *are the attacker's budget and the game-over threshold value, respectively.*

Remark. (i) Note that $V(T) = \{r, u_1, \ldots, u_n\}$, where r is the designated root that indicates the start of an attack. (ii) In most situations we have the weights c and p being non-negative rational numbers, and $p(r) = 0$.

Recall that in a rooted tree T each non-root vertex $u \in V(T)$ has exactly one parent. We let $e(u) \in E(T)$ denote the unique edge connecting u to its parent. For the root r, we let $e(r)$ be the empty set and $c(e(r))$ be 0. For a tree T with $u \in V(T)$, we let $T(u)$ denote the (largest) subtree of T rooted at u. It is easy to see the correspondence between Definitions 2.1 and 2.3. Analogously to Definition 2.2, we next define a *cyber-security attack strategy* in the weighted-tree model.

Definition 2.4. *A* cyber-security attack strategy (CSAS) *in a CSM presented as* $M = (T, c, p, B, G)$ *is given by a subtree* T' *of* T *that contains the root* r *of* T.

- *We define the* cost *of a CSAS* T' *to be* $c(T') = \sum_{u \in V(T')} c(e(u))$.
- *We define a* valid CSAS (VCSAS) *to be a CSAS* T' *with* $c(T') \leq B$.
- *We define the* prize *of a CSAS* T' *to be* $p(T') = \sum_{u \in V(T')} p(u)$.

A game-over attack strategy (GOAS) *in a CSM* $M = (T, c, p, B, G)$ *is a VCSAS* T' *with* $p(T') \geq G$. *We sometimes refer to such a GOAS simply as a* cyber-security attack *or* cyber attack *for short.*

Note that in Definition 2.4 we use c (resp. p) to denote the total cost (respectively, total prize) of a cyber-security attack strategy. We also use c (resp. p) as the penetration-cost weight function (respectively, target-acquisition-value weight function). The overloading of this notation should not cause any confusion. Throughout the remainder of the paper, we will use Definitions 2.3 and 2.4.

2.4 Cyber-Attack Problems in the Game-Over Model

We now state some natural questions based on the CSM.

Problem 2.1. GIVEN: *A cyber-security model* $M = (T, c, p, B, G)$.

- GAME-OVER ATTACK STRATEGY, DECISION PROBLEM (GOAS-DP): *Is there a game-over attack strategy in* M*?*
- GAME-OVER ATTACK STRATEGY, OPTIMIZATION PROBLEM (GOAS-OP): *What is the maximum prize of a valid game-over attack strategy in* M*?*

Needless to say, some special cases are also of interest, in particular, in Problems 2.1 when c is (i) a constant rational function, (ii) an integer-valued function, or (iii) takes only finitely many given rational values. We explore the general GOAS and these other questions in the following sections.

2.5 Some Limitations of the Model

Our model is a theoretical model. It is designed to give us a deeper understanding of cyber attacks and cyber-attack strategies. Of course, a real adversary is not in possession of complete knowledge about a system and its penetration costs. Nevertheless, it is interesting to suppose that an adversary is in possession of all of this information, and then to see what an adversary is capable of achieving under these circumstances. Certainly an adversary with less information could do no better than our fully informed adversary.

We are considering systems as they are. That is, we are given some system, targets, and penetration costs. If the system is a real system, we are not concerned about how to improve the security of that system per se. We assume that the system is already in a hardened state. We then examine how difficult it would be to attack such a system. We do not examine the question of implementations of a system. Our model can be used on any existing system. Some real systems will have more than one possible path to attack a target. And, in the future it may be worth generalizing the model to structures other than trees. The first step is to look at trees and derive some insight from these cases.

We have purposely chosen a target acquisition function which is simple. That is, we merely add together the total costs of the targets acquired. Studying this simple acquisition function is the first step. It may be interesting to study more-complex acquisition functions in the future. For example, one can imagine two targets that in and of themselves are of no real value, but when the information contained in the two are combined they are of great value. In some cases our additive function can capture this type of target depending on the structure of the model.

We describe the notion of a game-over component. In the model this concept is an abstract one. A set of components whose total value exceeds a given threshold comprise a "game-over component." A game-over component is not necessarily a single target although one can think of a high-cost target, which is included as a

target in a set of targets that push us over the game-over threshold, as being the game-over component.

For easy reference, the following table contains our most common abbreviations, their spelled out meaning, and where they are defined.

CSM	cyber-security (game-over) model	Def. 2.3
CSAS	cyber-security attack strategy	Def. 2.4
VCSAS	valid cyber-security attack strategy	Def. 2.4
GOAS	game-over attack strategy	Def. 2.4
GOAS-DP	game-over attack strategy, decision problem	Def. 2.1
GOAS-OP	game-over attack strategy, optimization problem	Def. 2.1

Table 2: Abbreviations we use throughout the paper, all defined in this section.

3 Complexity of Cyber-Attack Problems

In this section we show that the general game-over attack strategy problems are intractable, that is, highly unlikely to be amenable to polynomial-time solutions. Consider a cyber-security attack model M, where T is a star centered at r having n leaves $u_1, \ldots, u_n$. Since each cyber-security attack T' of M can be presented as a collection $E' \subseteq E(T)$ of edges of T, and hence also as a collection of vertices $V' \subseteq V(T)$ by $T' = T[\{r\} \cup V']$, and vice versa, each collection of vertices $V' \subseteq V(T)$ can be presented as $V' = V(T')$ for some cyber-security attack T' of M, and the GOAS-DP is exactly the decision problem of the 0/1-KNAPSACK PROBLEM [10], and the GOAS-OP is the optimization problem of the KNAPSACK PROBLEM. Note that the 0/1-KNAPSACK PROBLEM is usually stated using natural numbers as weights, but clearly the case for weights consisting of rational numbers is no easier to solve yet still in NP. So, we have the following observation.

Observation 3.1. *The* GOAS-DP *is NP-complete; the* GOAS-OP *is an NP-hard optimization problem.*

Remark. Observation 3.1 answers an open question in the last section of [15], where it is asked whether or not the LST-TREE PROBLEM can be solved in polynomial time (we presume) for general edge lengths. Observation 3.1 is similar to [7, Theorem 2], where also a star is considered to show that their SUBTREEE is as hard as KNAPSACK.

Notice that the NP-completeness of GOAS-DP is a double-edge sword. It suggests that even an attacker who has detailed knowledge of the defenses of a cyber-security system would find the problem of allocating his (attack) resources difficult. On the other hand, the NP-completeness also makes it difficult for the defender to assess the security of his system. However, we will see in Section 5, that if we allow a slight proportional increase of the attacker's budget B to an amount

of $(1+\epsilon)B$ for an $\epsilon \geq 0$, then GOAS-OP admits a polynomial time approximation scheme, so it can be solved in time polynomial in n and $1/\epsilon$.

Sections 4, 5, and 6 consider the complexity of cyber-security attacks where c is a constant-valued cost function, an integer-valued cost function, and a rational-valued cost function of finitely many possible values, respectively. In Section 5, as mentioned, we also obtain an approximation algorithm for solving GOAS-OP, and a solution on general weighted non-rooted trees. In all cases we are able to give reasonably efficient algorithms for solving GOAS-OP.

4 Cyber Attacks with Constant Penetration Costs

In this section we show that if all penetration costs have the same value then the GAME-OVER ATTACK STRATEGY PROBLEMS can be solved efficiently in polynomial time. Consider a CSM M, where c is a constant function taking a constant rational value $c(e) = c$ for each $e \in E(T)$. That is, all penetration costs are a fixed-rational value. This variant is the first interesting case of the GOAS-DP and GOAS-OP, as there are related problems and solutions in the literature. One of the first papers on maximum-weight subtrees of a given tree with a specific root is [1], where it is shown that the *rooted subtree problem*, that is, to find a maximum-weight subtree with a specific root from a given set of subtrees, is in polynomial time if, and only if, the *subtree packing problem*, that is, to find maximum-weight packing of vertex-disjoint subtrees from a given set of subtrees (where the value of each subtree can depend on the root), is in polynomial time. In more-recent papers the *weight-constrained maximum-density subtree problem (WMSP)* is considered: given a tree T having n vertices, and two functions $l, w : E(T) \to \mathbb{Q}$ representing the "length" and "weight" of the edges, respectively, determine the subtree T' of T such that $\sum_{e \in E(T')} w(e) / \sum_{e \in E(T')} l(e)$ is a maximum, subject to $\sum_{e \in E(T')} w(e)$ having a given upper bound. In [13] an $O(w_{\max} n)$-time algorithm is given to solve the related, and more restricted, *weight-constrained maximum-density path problem (WMPP)*, as well as an $O(w_{\max}^2 n)$-time algorithm to solve the WMSP. In [15] an $O(nU^2)$-time algorithm is given for the WMSP, where U is the maximum total length of the subtree, and in [22] an $O(nU \lg n)$-time algorithm for the WMSP is given, which is an improvement in the case when $U = \Omega(\lg n)$. The WMSP has a wide range of practical applications. In particular, the related WMPP has applications in computational biology [13], and the related *weight-constrained least-density path problem (WLPP)* also has applications in computational biology, as well as in computer, traffic, and logistic network designs [15].

The WMSP is similar to our problem, and some of the same approaches used in [13], [15], and [22] can be applied in our case, namely the techniques of recursion and dynamic programming. There are not existing results that apply directly to our problems. Note that there is a subtle difference between our GOAS-OP and the WMSP, as a maximum-weight subtree (that is, with the prize $p(T')$ a maximum) might have low density and vice versa; a subtree of high density might be "small" with low total weight (that is, prize).

In [7] a problem on trees related to the Traveling Salesman Problem with profits is studied, which is similar to what we do. Both here and in [7] the most general form of the problems considered, in our case GOAS-DP in Observation 3.1 and in their case (as mentioned above) SUBTREEE in [7, Theorem 2], are observed to be as hard as KNAPSACK and hence NP-complete. Also, the results of fixed costs, in our case Theorem 4.1 and in their case [7, Theorem 3], the problems are shown to be solvable in $O(n)$ time, given certain conditions. Theorem 4.1, however, provides a precise accounting for the time complexity and for certain values of m, defined there, our algorithm would be faster than that given in [7]. Their work is not in the context of cyber-security, and does not handle cases as general as this work.

For a CSM M, where c is a constant function, we first note that T' is a VCSAS if and only if $m = |E(T')| \leq \lfloor B/c \rfloor$. Hence, in this case the GOAS-OP reduces to finding a CSAS T' with at most m edges having $p(T')$ at a maximum. Note that if $m \geq n$, then the GOAS-OP is trivial since $T' = T$ is the optimal subtree. Hence, we will assume the budget B is such that $m < n$.

In what follows, we will describe our dynamic programming setup to solve GOAS-OP in this case. The core of the idea is simple: we construct a $d(u) \times (m+1)$ matrix for each vertex u in the tree T that stores the maximum prize of a subtree rooted at u on at most k edges and that contains only the rightmost $d(u) - i + 1$ branches from u, for each $k \in \{0, 1, \ldots, m\}$ and $i \in \{1, \ldots, d(u)\}$.

More specifically, we proceed as follows. We may assume that our rooted tree T has its vertices ordered from left-to-right in some arbitrary but fixed order, that is, T is a *planted plane tree*. Since T has $n \geq 1$ non-root vertices and $n+1$ vertices total, we know by a classic counting exercise [2] that the number of planted plane trees on $n+1$ vertices is given by the Catalan numbers C_n by obtaining a defining recursion for C_n by decomposing each planted plane tree into two rooted subtrees. Using this decomposition, we introduce some notation. For a subtree τ of T rooted at $u \in V(T)$ denote by $\tau(v)$ the largest subtree of τ that is rooted at a vertex v (if $v \in T[V(\tau)]$). Denote by u_ℓ the leftmost child of u in τ (if it exists). Let $\tau_\ell = \tau(u_\ell)$ denote the subtree of τ generated by u_ℓ, that is, the largest subtree of T rooted at u_ℓ. Finally, let $\tau'' = \tau - V(\tau_\ell) = T[V(\tau) \setminus V(\tau_\ell)]$ denote the subtree of τ generated by the vertices not in τ_ℓ. In this way we obtain a decomposition/partition of the planted plane tree τ into two vertex-disjoint subtrees τ_ℓ and τ'' whose roots are connected by a single edge $e(u_\ell)$. In particular, for each vertex $u \in V(T)$, we have a partition of $T(u)$ into $T(u)_\ell = T(u_\ell)$ and $T(u)''$, which we will denote by $T''(u)$ (that is $T(u)'' = T''(u)$). Note that if u is a leaf, then $T(u) = T''(u) = \{u\}$ and $u_\ell = T(u_\ell) = \emptyset$. Also, if u has exactly one child, which therefore is its leftmost child u_ℓ, then $T(u)$ is the two-path between u and its only child u_ℓ, $T''(u) = \{u\}$, and $T(u_\ell) = \{u_\ell\}$. Assuming the degree of u is $d(u)$, we can recursively define the trees $T^1(u), \ldots, T^{d(u)}(u)$ by

$$\begin{aligned} T^1(u) &= T(u), \\ T^{i+1}(u) &= (T^i)''(u). \end{aligned}$$

For each vertex $u \in V(T)$, we create a $d(u) \times (m+1)$ rational matrix as follows:

$$\mathbf{M}(u) = \begin{bmatrix} M_0^1(u) & M_1^1(u) & \cdots & M_m^1(u) \\ M_0^2(u) & M_1^2(u) & \cdots & M_m^2(u) \\ & & \vdots & \\ M_0^{d(u)}(u) & M_1^{d(u)}(u) & \cdots & M_m^{d(u)}(u) \end{bmatrix},$$

where $M_k^i(u)$ is the maximum prize of a subtree of $T^i(u)$ rooted at u with at most k edges for each $i \in \{1, \ldots, d(u)\}$ and $k \in \{0, 1, \ldots, m\}$. In particular, $M_0^i(u) = p(u)$ for each vertex u and $i \in \{1, \ldots, d(u)\}$. For each leaf u of T, and each i and k, we set $M_k^i(u) = p(u)$, and for each internal vertex u we have a recursion given in the following way: for a vertex u and an *arbitrary* subtree τ rooted at u, we let $M_k(u;\tau)$ be the maximum prize of a subtree of τ rooted at u having k edges or 0 if vertex u does not exist. If a maximum-prize subtree of τ with k edges does not contain the edge from u to its leftmost child u_ℓ, then $M_k(u;\tau) = M_k(u;\tau'')$. Otherwise, such a maximum subtree contains $i-1$ edges from τ_ℓ and $k-i$ edges from τ''. The following lemma is easy to show.

Lemma 4.1. *The arbitrary subtree τ rooted at u is a maximum-prize subtree with at most k edges that contains the leftmost child u_ℓ of u if and only if the included subtree of τ_ℓ is a maximum-prize subtree with at most $i-1$ edges rooted at u_ℓ and the included subtree of τ'' is a maximum-prize subtree with at most $k-i$ edges rooted at u for some $i \in \{1, \ldots, k\}$.*

By Lemma 4.1 we therefore have the following recursion:

$$M_k(u;\tau) = \max\left(M_k(u;\tau''), \max_{1 \le i \le k} \left(M_{i-1}(u_\ell;\tau_\ell) + M_{k-i}(u;\tau'') \right) \right). \tag{1}$$

Since now $M_k^i(u) = M_k(u;T^i(u))$ for each i and k, we see that we can compute each $M_k^i(u)$ from the smaller M's as given in (1) using $O(k)$-arithmetic operations. Because $k \in \{0, 1, \ldots, m\}$, this fact means in $O(m)$-arithmetic operations. Since we assume each arithmetic operation takes one step, we have that each $M_k^i(u)$ can be computed in $O(m)$-time given the required inputs. Therefore, $\mathbf{M}(u)$ can be computed in $d(u)m \cdot O(m) = d(u)O(m^2)$-time. Performing these calculations for each of the n vertices of our given tree T, we obtain by the Handshaking Lemma a total time of

$$t(n) = \sum_{u \in V(T)} d(u)O(m^2) = O(m^2) \sum_{u \in V(T)} d(u) = O(m^2)2(n-1) = O(m^2 n).$$

We finally compute a maximum prize VCSAS T' in M by $p(T') = M_m^1(r)$ for the root r of T. We conclude by the following theorem.

Theorem 4.1. *If $M = (T, c, p, B, G)$ is a CSM, where T has n vertices, c is a constant function, and $m = \lfloor B/c \rfloor$ then the* GOAS-OP *can be solved in $O(m^2 n)$-time.*

Remark. (i) Note that Theorem 4.1 is similar to [7, Theorem 3]. (ii) Also note that the overhead constant is "small": for each vertex u, each k, and each i by (1) each of $M_k^i(u) = M_k(u; T^i(u))$ uses exactly $2k$ arithmetic operations, namely k additions and k comparisons. Hence, the exact number of arithmetic operations can, by the Handshaking Lemma, be given by

$$N(n,m) = \sum_{u \in V(T)} \sum_{k=0}^{m} d(u)(2k) = \sum_{u \in V(T)} d(u) \sum_{k=0}^{m} 2k = 2|E(T)|m^2 = 2(n-1)m^2.$$

We obtain an overhead constant of two. Since we assumed the budget given is such that $m < n$, we see that the GOAS-OP can be solved in $O(n^3)$ time.

Corollary 4.1. *The* GOAS-DP *when restricted to constant-valued penetration costs can be solved in $O(n^3)$ time and is in P.*

5 Cyber Attacks with Integer Penetration Costs and an Approximation Scheme

In this section we show that if all penetration costs are non-negative integers then the GAME-OVER ATTACK STRATEGY PROBLEMS can be solved in pseudo-polynomial time. We will then use that to obtain a polynomial time approximation algorithm.

5.1 Integer valued cost

Consider now a CSM $M = (T, c, p, B, G)$, where c is a non-negative integer-valued function, that is, $c(e) \in \{0, 1, 2, \ldots\}$ for each $e \in E(T)$. Note that we can contract T by each edge e with $c(e) = 0$, thereby obtaining a tree for our CSM M, where $c : E(T) \to \mathbb{N}$ takes only positive-integer values. We derive a polynomial-time algorithm in terms of n and B to solve the GOAS-OP. We can assume B is an integer here as well since otherwise we could just replace B with $\lfloor B \rfloor$. To produce our new algorithm we will tweak the argument given in Section 4 for the case when the cost function c is a constant.

Using the same decomposition of a subtree τ of T into u_ℓ and τ'' for our dynamic programming scheme, for each vertex u we will assign, as before, a $d(u) \times (B+1)$ integer matrix as follows:

$$\mathbf{N}(u) = \begin{bmatrix} N_0^1(u) & N_1^1(u) & \cdots & N_B^1(u) \\ N_0^2(u) & N_1^2(u) & \cdots & N_B^2(u) \\ & & \vdots & \\ N_0^{d(u)}(u) & N_1^{d(u)}(u) & \cdots & N_B^{d(u)}(u) \end{bmatrix},$$

where $N_k^i(u)$ is the maximum prize of a subtree of $T^i(u)$ rooted at u of total cost at most k for each $i \in \{1, \ldots, d(u)\}$ and $k \in \{0, \ldots, B\}$. As before, we have $N_0^i(u) = p(u)$ for each vertex u. Similarly to Lemma 4.1, we obtain the following.

Lemma 5.1. *The arbitrary subtree τ rooted at u is a maximum-prize subtree of total cost at most k that contains the leftmost child u_ℓ of u if and only if the included subtree of τ_ℓ is a maximum-prize subtree of total cost at most $i - c(e(u_\ell))$ rooted at u_ℓ and the included subtree of τ'' is a maximum-prize subtree of total cost $k - i$ rooted at u, for some $i \in \{c(e(u_\ell)), \ldots, k\}$.*

Using similar notation and definitions as in Section 4, by Lemma 5.1 we get the following recursion:

$$N_k(u;\tau) = \max\left(N_k(u;\tau''), \max_{c(e(u_\ell))\le i\le k}\left(N_{i-c(e(u_\ell))}(u_\ell;\tau_\ell) + N_{k-i}(u;\tau'')\right)\right), \quad (2)$$

and we obtain similarly the following.

Theorem 5.1. *If $M = (T, c, p, B, G)$ is a CSM, where T has n vertices and $c : E(T) \to \mathbb{N}$ takes only positive-integer values, then the* GOAS-OP *can be solved in $O(B^2 n)$-time.*

Remark. (i) Although we are not able to obtain a compact expression for the exact number of arithmetic operations that yield Theorem 5.1, the bound $N(n, B) = 2(n-1)B^2$ still is an upper bound, as for Theorem 4.1. (ii) Note the assumption that c is an *integer*-valued cost function is crucial, since otherwise, we would not have been able to use the recursion (2) in at most B steps.

Corollary 5.1. *The* GOAS-DP *when restricted to integer-valued penetration costs can be solved in pseudo-polynomial time.*

5.2 Approximation Scheme

We now can present a polynomial time approximation scheme (PTAS) for solving the GOAS-OP from Problem 2.1. In Observation 3.1 we saw that the GOAS-OP is an NP-hard optimization problem. But this is not the whole story; although it is hard to compute the exact solution, one can obtain a polynomial time approximation algorithm if we allow slightly more budget for the attacker than he/she wants to spend. We will in this section describe one such approximation scheme. Our approach here is similar to the PTAS for the optimization of the 0/1-KNAPSACK PROBLEM presented in the classic text [16, Section 17.3].

We saw in Theorem 5.1 that GOAS-OP can be solved in $O(B^2 n)$-time, if the cost is integer valued and B is the budget of the attacker. So for large B this can be far polynomial time. For each fixed $t \in \mathbb{N}$ we can write the integer cost $c(e)$ of each edge $e \in E(T)$ as

$$c(e) = c_q(e) + c_r(e), \quad \text{where } c_r(e) = c(e) \bmod 2^t, \quad (3)$$

that is, we obtain a new cost function c_q by ignoring the last t digits of $c(e)$ when it is written as a binary number. Since each c_q is divisible by 2^t, solving GOAS-OP for c_q and budget B is the same as solving it for the cost function $2^{-t}c_q$ and budget

$2^{-t}B$. Therefore, we can by Theorem 5.1 solve the GOAS-OP for this new cost function c_q in $O((2^{-t}B)^2 n)$-time.

Let T' (resp. T'_q) be an optimal GOAS-OP subtree of T w.r.t the cost c (resp. c_q), so $p(T')$ is maximum among subtrees with c-weight $\leq B$, and $p(T'_q)$ is maximum among subtrees with c_q-weight $\leq B$. In this case we have

$$c(T'_q) = c_q(T'_q) + c_r(T'_q) \leq B + |E(T'_q)| \cdot 2^t \leq B + n2^t. \tag{4}$$

Also, since $c_q(T') \leq c(T') \leq B$ we have by the definitions of T' and T'_q that $p(T') \leq p(T'_q)$. Therefore if there is a GOAS T' w.r.t. the cost c, then there certainly is one w.r.t. the cost c_q, namely T_q. Hence, if $\epsilon = \frac{n2^t}{B}$, then we obtain from (4) that $c(T_q) \leq (1+\epsilon)B$ and T'_q is here definitely a GOAS that further can be computed in $O((n/\epsilon)^2 n) = O((1/\epsilon)^2 n^3)$-time. Conversely, for a given $\epsilon \geq 0$, we obtain such an approximation algorithm by considering the cost c_q defined by (3) where

$$t = \left\lfloor \lg\left(\frac{\epsilon B}{n}\right) \right\rfloor. \tag{5}$$

We therefore have the following.

Theorem 5.2. *The* GOAS-OP *admits a polynomial time approximation scheme; for every $\epsilon \geq 0$ a GOAS T' of cost of at most $(1+\epsilon)B$ can be computed in $O((1/\epsilon)^2 n^3)$-time.*

Remark. (i) In establishing the above Theorem 5.2 we started with an integer cost function $c : E(T) \to \mathbb{N}$. The same approach could have been used for a rational cost function $c : E(T) \to \mathbb{Q}$ where $c(e)$ has d binary binary digits after its binary point (i.e. radix point when written as a rational number in base 2.) By considering a new integer valued cost function $c' : E(T) \to \mathbb{N}$, where $c'(e) = 2^d c(e)$ for each $e \in E(T)$, we can in the same manner as used above, obtain an approximation algorithm where we replace B with $B' = 2^d B$. Needless to say however, in this case the corresponding cost function c'_q is obtained by truncating or ignoring only $t - d$ of the digits of c' (instead of the t digits of c), to obtain a solution using a budged of $(1+\epsilon)B$. (ii) Further along these lines, if the cost function $c : E(T) \to \mathbb{Q}$ is given as a fraction $c(e) = a(e)/b(e)$, where $a(e), b(e) \in \mathbb{N}$ are relatively prime, we can let M be the least common multiple of the $b(e)$ where $e \in E(T)$ and obtain by scaling by M a new integer valued cost function $c'' : E(T) \to \mathbb{N}$ where $c''(e) = Mc(e)$ for each $e \in E(T)$. Again, since c'' is integer valued we can in the same manner obtain an approximation algorithm where we replace B with $B'' = MB$. In this case the corresponding cost function c''_q is obtained by truncating or ignoring even fewer digits, namely $t - \lg M$ of the digits of c''. This will also yield a polynomial time approximation algorithm in terms of n and $1/\epsilon$ despite the fact that M can become very large (i.e. if all the costs have pairwise relatively prime denominators $b(e)$.)

5.3 General Weighted Trees

In our framework a CSM M is presented as a rooted tree provided with two weight functions: one on the vertices and one on the edges. In the model the root serves merely as a starting vertex and does not (usually) carry any weight (that is, has no prize attached to it). However, given a general non-rooted tree T provided with two edge-weight functions $w, w' : E(t) \to \mathbb{Q}$, we can always add a root to some vertex and then push the weights of one of the weight functions, say w down to the unique vertex away from the root. In this way we obtain a CSM M to which we can apply both Theorems 4.1 and 5.1. With this slight modification, we have the following corollary for general weighted trees.

Corollary 5.2. *Let T be a tree on n vertices, $w, w' : E(T) \to \mathbb{Q}$ two edge-weight functions, and B, G two rational numbers. If the function w is either (i) a rational constant $c \in \mathbb{Q}$ or (ii) integer-valued, then the existence of a subtree T' of T such that $w'(T') \leq B$ and $w(T')$ is a maximum can be determined in $O(m^2 n)$-time, where $m = \lfloor B/c \rfloor$ in case (i), and in $O(B^2 n)$-time in case (ii).*

6 Cyber Attack with Rational Penetration Costs

In this section we consider the more-general case of a CSM $M = (T, c, p, B, G)$ where the cost function $c : E(T) \to \mathbb{Q}$ takes at most d distinct rational values, say $c_1, \ldots, c_d \in \mathbb{Q}$. This case can model quite realistic scenarios, as there are currently only a finite number of known encryption methods and cyber-security designs, where a successful hack for each method/design has a specific penetration cost. As in previous sections, we will utilize dynamic programming and recursion based on the splitting of a subtree τ of a planted plane subtree into two subtrees τ_ℓ and τ'' as in (1) and (2). However, here we are dealing with rational-cost values (i.e. arbitrary *real* values from all practical purposes), and that the we are able to obtain a polynomial time procedure in this case is not as direct.

Note that if M is the least common multiple of all the denominators of $c_1, \ldots, c_d$, then by multiplying the cost and the budget of the attacker through by M, we obtain an integer valued cost function Mc, which then can by Theorem 5.1 be solved pseudo polynomially in $O(M^2 B^2 n)$-time. Our goal here in this section, however, is to develop an algorithm to solve GOAS-OP in time polynomial in n alone.

For each $i \in \{1, \ldots, d\}$, let $n_i = |\{e \in E(T) : c(e) = c_i\}|$, and so $\sum_{i=1}^{d} n_i = n = |E(T)| = |V(T)| - 1$. Let $\mathcal{B} = \{0, 1, \ldots, n_1\} \times \cdots \times \{0, 1, \ldots, n_d\} \subseteq \mathbb{Z}^d$, and note that $|\mathcal{B}| = \prod_{i=1}^{d}(n_i + 1)$. Denote a general d-tuple of $\mathbb{Q}^d$ by $\tilde{x} = (x_1, \ldots, x_d)$, and let $\tilde{x} \leq \tilde{y}$ denote the usual component-wise partial order $x_i \leq y_i$, for each $i \in \{1, \ldots, d\}$. If $\tilde{c} = (c_1, \ldots, c_d) \in \mathbb{Q}^d$ is the *rational-cost vector*, let $\mathcal{C} = \{\tilde{x} \in \mathbb{Q}^d : \tilde{x} \geq \tilde{0},\ \tilde{c} \cdot \tilde{x} \leq B\} \subseteq \mathbb{Q}^d$ denote the d-dimensional pyramid in $\mathbb{Q}^d$ with the $d + 1$ vertices given by the origin $\tilde{0} = (0, \ldots, 0)$ and $(0, \ldots, B/c_i, \ldots, 0)$, where $i \in \{1, \ldots, d\}$. To estimate the number of non-negative integral points in $\mathcal{C}$, we count the number of unit d-cubes within the pyramid $\mathcal{C}$. Since $\lfloor x \rfloor \leq x \leq \lfloor x \rfloor + 1$

for each rational x, then each $\tilde{x} \in \mathcal{C}$ is contained in the unit d-cube with the line segment from $\lfloor \tilde{x} \rfloor = (\lfloor x_1 \rfloor, \ldots, \lfloor x_d \rfloor)$ to $\lfloor \tilde{x} \rfloor + \tilde{1} = (\lfloor x_1 \rfloor + 1, \ldots, \lfloor x_d \rfloor + 1)$ as its diagonal. Since $\tilde{c} \cdot \tilde{x} \leq B$, then $\tilde{c} \cdot (\lfloor \tilde{x} \rfloor + \tilde{1}) \leq B + \sum_{i=1}^{d} c_i$, and hence, the number of integral points in $\mathcal{C}$ is at most the volume $\mathcal{V}(\mathcal{C}')$ of the associated pyramid $\mathcal{C}' = \{\tilde{x} \in \mathbb{Q}^d : \tilde{x} \geq \tilde{0},\ \tilde{c} \cdot \tilde{x} \leq B'\} \subseteq \mathbb{Q}^d$, where $B' = B + \sum_{i=1}^{d} c_i$, that is, at most $\lfloor \mathcal{V}(\mathcal{C}') \rfloor$, where

$$\mathcal{V}(\mathcal{C}') = \frac{1}{d!} \prod_{i=1}^{d} \frac{B'}{c_i} = \frac{1}{d!} \prod_{i=1}^{d} \left(\frac{B + \sum_{j=1}^{d} c_j}{c_i} \right).$$

Note that a CSAS T' of a CSM M has k_i edges of cost c_i for each i if and only if $\tilde{k} \in \mathcal{B} \cap \mathcal{C}'$.

Definition 6.1. *For each i let $m_i = \min(\lceil B'/c_i \rceil, n_i)$, and let $m = \sum_{i=1}^{d} m_i$.*

Remark. Note that we have $m = \sum_{i=1}^{d} m_i \leq \sum_{i=1}^{d} n_i = n$, and therefore any upper bound polynomial in m will yield a bound in the same polynomial in terms of n.

If $\mathcal{C}'' = \{0, 1, \ldots, \lceil B'/c_1 \rceil\} \times \cdots \times \{0, 1, \ldots, \lceil B'/c_d \rceil\}$, then $\mathcal{C}' \cap \mathbb{Z}^d \subseteq \mathcal{C}''$, and

$$\mathcal{B} \cap \mathcal{C}' = \mathcal{B} \cap (\mathcal{C}' \cap \mathbb{Z}^d) \subseteq \mathcal{B} \cap \mathcal{C}'' = \{0, 1, \ldots, m_1\} \times \cdots \times \{0, 1, \ldots, m_d\} \qquad (6)$$

Hence, by the Inequality of Arithmetic and Geometric Mean (IAGM), we get

$$|\mathcal{B} \cap \mathcal{C}'| \leq |\mathcal{B} \cap \mathcal{C}''| = \prod_{i=1}^{d} (m_i + 1) \leq \left(\frac{\sum_{i=1}^{d} (m_i + 1)}{d} \right)^d = \left(\frac{m}{d} + 1 \right)^d.$$

We summarize in the following.

Observation 6.1. *If M is a CSM with n_i edges of cost c_i for each $i \in \{1, \ldots, d\}$, then $|\mathcal{B} \cap \mathcal{C}'| \leq (m/d + 1)^d$, which is a polynomial in $m = \sum_{i=1}^{d} m_i$ of degree d.*

Remark. Note that if $B'/c_i \leq n_i$ for each i, then $m_i = \min(\lceil B'/c_i \rceil, n_i) = \lceil B'/c_i \rceil$. In this case we have $\mathcal{C}' \cap \mathbb{Z}^d \subseteq \mathcal{B}$ and so $\mathcal{C}' \cap \mathbb{Z}^d = \mathcal{C}' \cap \mathbb{Z}^d \cap \mathcal{B} = \mathcal{C}' \cap \mathcal{B}$, and so again by the IAGM, we obtain

$$|\mathcal{B} \cap \mathcal{C}'| = |\mathcal{C}' \cap \mathbb{Z}^d| \leq \lfloor \mathcal{V}(\mathcal{C}') \rfloor = \left\lfloor \frac{1}{d!} \prod_{i=1}^{d} (m_i + 1) \right\rfloor \leq \left\lfloor \frac{1}{d!} \left(\frac{m}{d} + 1 \right)^d \right\rfloor,$$

where now $m = \sum_{i=1}^{d} \lceil B'/c_i \rceil$, which shows that, although polynomial in m of the same degree d as in Observation 6.1, the number of possible $\tilde{k} \in \mathcal{B} \cap \mathcal{C}'$ is a much smaller fraction of $(m/d + 1)^d$.

We now proceed with our setup for our dynamic programming scheme. As before, the idea is simple; we construct a multi-dimensional matrix/array for each

vertex u of T, the construction of which is computed in a recursive manner, as for the previous 2×2 matrices $\mathbf{M}(u)$ and $\mathbf{N}(u)$.

Specifically, for each vertex u we assign a $d(u) \times |\mathcal{B} \cap C'|$-fold array

$$\mathbf{A}(u) = \left[A^i_{\tilde{k}}(u)\right]_{\tilde{k} \in \mathcal{B} \cap C',\ 1 \le i \le d(u)},$$

where $A^i_{\tilde{k}}(u)$ is the maximum prize of a subtree of $T^i(u)$ containing k_j edges of cost c_j for each $j \in \{1, \ldots, d\}$ and each $\tilde{k} \in \mathcal{B} \cap C'$. For $\tilde{0} = (0, \ldots, 0)$, we have $A^i_{\tilde{0}}(u) = p(u)$ for each vertex u for $i = 1, \ldots, d(u)$.

CONVENTION: For $i \in \{1, \ldots, d\}$ and an edge $e \in E(T)$, let $\delta_i(e) = \delta^{c(e)}_{c_i}$, where for every pair of rational numbers $x, y \in \mathbb{Q}$

$$\delta^y_x = \begin{cases} 1 & \text{if } x = y, \\ 0 & \text{otherwise} \end{cases}$$

is the *Kronecker delta function.* Further, let $\tilde{\delta}(e) = (\delta_1(e), \ldots, \delta_d(e))$.

As in (1) and (2), we use the same decomposition of a subtree τ of T into τ_ℓ and τ'', and as with previous Lemmas 4.1 and 5.1, we have the following.

Lemma 6.1. *The subtree τ rooted at u is a maximum-prize subtree among those with k_i edges of cost c_i for each i and that contains the leftmost child u_ℓ of u if and only if the included subtree of τ_ℓ is a maximum-prize subtree among those rooted at u_ℓ and with α_i edges of cost c_i for each i and the included subtree of τ'' is a maximum-prize subtree rooted at u among those that do not contain u_ℓ and with β_i edges of cost c_i for each i, for some $\tilde{\alpha}, \tilde{\beta} \in \mathcal{B} \cap C'$, where $\tilde{\alpha} + \tilde{\beta} = \tilde{k} - \tilde{\delta}(e(u_\ell))$.*

For a vertex u and an arbitrary subtree τ rooted at u, we let $A_{\tilde{k}}(u; \tau)$ be the maximum prize of a subtree of τ rooted at u with k_i edges of cost c_i for each $i \in \{1, \ldots, d\}$. If a maximum-prize subtree of τ with k_i edges of cost c_i does not contain the edge from u to its leftmost child u_ℓ, then $A_{\tilde{k}}(u; \tau) = A_{\tilde{k}}(u; \tau'')$. Otherwise, such a maximum subtree contains α_i edges of cost c_i from τ_ℓ and β_i edges of cost c_i from τ'', where $\alpha_i + \beta_i = c_i - \delta(e(u_\ell))$ for each $i \in \{1, \ldots, d\}$. Finally, for each leaf u of T, each i, and $\tilde{k} \in \mathcal{B} \cap C'$; we set $A^i_{\tilde{k}}(u) = p(u)$. As previously, we get by Lemma 6.1 the following recursion.

$$A_{\tilde{k}}(u; \tau) = \max\left(A_{\tilde{k}}(u; \tau''), \max_{\tilde{\alpha} + \tilde{\beta} = \tilde{k} - \tilde{\delta}(e(u_\ell))} \left(A_{\tilde{\alpha}}(u_\ell; \tau_\ell) + A_{\tilde{\beta}}(u; \tau'')\right)\right). \quad (7)$$

Lemma 6.2. *The evaluation of each $A^i_{\tilde{k}}(u)$ takes at most $2(m/d + 1)^d$ arithmetic operations.*

Proof. For each $\tilde{x} = (x_1, \ldots, x_d) \in \mathbb{Q}^d$, let $\pi^+(\tilde{x}) = \prod_{i=1}^d (x_i + 1)$. By (7) each $A^i_{\tilde{k}}(u)$ requires $\pi^+(\tilde{k} - \tilde{\delta}(e(u_\ell)))$ additions and $\pi^+(\tilde{k} - \tilde{\delta}(e(u_\ell)))$ comparisons, and hence all in all $2\pi^+(\tilde{k} - \tilde{\delta}(e(u_\ell)))$ arithmetic operations.

By (6) we have that $\tilde{k} \in \mathcal{B} \cap C' \subseteq \mathcal{B} \cap C''$, and hence, $k_j \leq m_j$ for each $j \in \{1, \ldots, d\}$. Thus, by the IAGM, there are at most

$$2\pi^+(\tilde{k} - \tilde{\delta}(e(u_\ell))) < 2\prod_{j=1}^{d}(k_j + 1) \leq 2\prod_{j=1}^{d}(m_j + 1) \leq 2\left(\frac{m}{d} + 1\right)^d$$

arithmetic operations for evaluating each $A_{\tilde{k}}^i(u)$. □

Assuming each arithmetic operation takes one step, the total running time to evaluate the entire array $\mathbf{A}(u)$ is at most a constant multiple of

$$\begin{aligned} N_d(n) &= \sum_{u \in V(T)} \sum_{\tilde{k} \in \mathcal{B} \cap C'} \sum_{i=1}^{d(u)} 2\left(\frac{m}{d} + 1\right)^d \\ &= \left(\sum_{u \in V(T)} d(u)\right)\left(\sum_{\tilde{k} \in \mathcal{B} \cap C'} 2\left(\frac{m}{d} + 1\right)^d\right) \\ &\leq 2|E(T)|\left(\frac{m}{d} + 1\right)^d 2\left(\frac{m}{d} + 1\right)^d \\ &= 4(n-1)\left(\frac{m}{d} + 1\right)^{2d}. \end{aligned}$$

We then obtain the desired maximum prize $p(T')$ of a VCSAS T' by $p(T') = \max_{\tilde{k} \in \mathcal{B} \cap C'}\left(A_{\tilde{k}}^1(r)\right)$ for the root r of T of our CSM M, which takes at most $|\mathcal{B} \cap C'| - 1 < (m/d + 1)^d$ comparisons. Hence, we obtain the following.

Theorem 6.1. *If $M = (T, c, p, B, G)$ is a CSM where T has n vertices, m is given by Definition 6.1, and $c : E(T) \to \mathbb{Q}$ takes at most d distinct rational values, then the* GOAS-OP *can be solved in $O(m^{2d}n)$-time.*

Remark. (i) Note that when $d = 1$, and hence $c_1 = c$, then m in Theorem 6.1 is given by $m = m_1 = \min(\lceil B'/c_1 \rceil, n) = \min(\lceil B/c \rceil + 1, n)$, whereas in Theorem 4.1 $m = \lceil B/c \rceil = \min(\lceil B/c \rceil, n)$, by the assumption that $\lceil B/c \rceil \leq n$. Still, the complexity when $d = 1$ in Theorem 6.1 clearly agrees with the complexity of $O(m^2 n)$ for solving the GOAS-OP when c is a constant function in Theorem 4.1. (ii) If each $m_i = O(f(n))$, for some "slow-growing" function of n, then Theorem 6.1 yields an $O(nf(n)^{2d})$-time algorithm for solving the GOAS-OP. In particular, if each $m_i = O(1)$, then Theorem 6.1 yields a linear-time in n algorithm to solve the GOAS-OP.

Corollary 6.1. *The* GOAS-DP *when restricted to d rational-valued penetration costs can be solved in polynomial time.*

7 Summary and Conclusions

This paper defined a new cyber-security model that models systems which are designed based on defense-in-depth. We showed that natural problems based on the

model were intractable. We then proved that restricted versions of the problems had either polynomial time or pseudo-polynomial time algorithms. Table 1 in Section 1 summarizes our results. They suggest that in a real system the penetration costs should vary, that is, although each level should be difficult to attack, the cost of breaking into some levels should be even higher. The tree representation of the models suggests that systems should be designed to distribute targets in a bushy tree, rather than in a narrow tree. Most security systems are linear, and such systems could be strengthen by distributing targets more widely, providing *defense-in-deception.* Although in most situations a cyber attacker will not a priori know exact penetration costs, target locations, and prizes, the model still gives us insight into which types of security designs would be more effective.

We conclude the paper with a number of open questions.

1. Can we quantify how much targets need to be distributed in order to maximize security? For example, does an $(n+1)$-ary tree provide provably better security than an n-ary tree?

2. Can we prove mathematically that the intuition of storing high-value targets deeper in the system and having higher penetration costs on the outer-most layers of the system results in the best security?

3. If targets are allowed to be repositioned periodically, what does that do to the complexity of the problems, and what is the best movement strategy for protecting targets?

4. Using the model, can one develop a set of benchmarks to rank the security of a particular system? How would one model prizes in a system?

5. Can the notion of time and intrusion detection be built into the model? That is, if an attacker tries to break into a certain container, the attacker may be locked out, resulting in game-over for that attacker, or perhaps may face an even higher new penetration cost.

6. Are there online variants of the model that are interesting to study? For example, a version where the topology of the graph changes dynamically or where only a partial description is known to the attacker.

Acknowledgments

This work was in part motivated by a talk that Bill Neugent of MITRE Corporation gave at the United States Naval Academy in the fall of 2011. We thank Bill for initial discussions about game-over issues relating to cyber-security models. Thanks also to Richard Chang for discussions about the model. – Finally, we like to thank the two anonymous referees for their careful reading of the paper, their pointed comments and suggestions which resulted in a greatly improved presentation of the results and made them more complete.

References

[1] El Houssaine Aghezzaf, Thomas L. Magnanti, and Laurence A. Wolsey. Optimizing Constrained Subtrees of Trees. *Mathematical Programming*, **71(2)**:113–126, Series A, (1995).

[2] Geir Agnarsson and Raymond Greenlaw. *Graph Theory: Modeling, Applications, and Algorithms*, Pearson Prentice Hall, Upper Saddle River, NJ, (2007).

[3] Robert C. Armstrong, Jackson R. Mayo, and Frank Siebenlist. Complexity Science Challenges in Cybersecurity, *Sandia Report*, March 2009.

[4] Tania Branigan. "Chinese Army to Target Cyber War Threat." *The Guardian (London)*. www.theguardian.com/world/2010/jul/22/chinese-army-cyber-war-department, retrieved October 1, 2013.

[5] Hayes Brown. "No Longer in the Shadows, Cyberwar's Potential is now an Open Secret." *Think Progress*. thinkprogress.org/security/2013/10/04/2699361/cyber-conflict-just-over-the-horizon/, retrieved October 15, 2013.

[6] Deepayan Chakrabarti and Christos Faloutsos. Graph Mining: Laws, Generators, and Algorithms. *ACM Computing Surveys*, **38(1)**, article 2, 69 pages, (2006).

[7] Sofie Coene, Carlo Filippi, Frits Spieksma, and Elisa Stevanato. Balancing Profits and Costs on Trees. *Networks*, **61(3)**:200–11, (2013).

[8] "2012 Cost of Cyber Crime Study: United States," *Ponemon Institute*, research report, 29 pages, October 2012.

[9] Daniel M. Dunlavy, Bruce Hendrickson, and Tamara G. Kolda. Mathematical Challenges in Cybersecurity. *Sandia Report*, February 2009.

[10] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, New York, (1979).

[11] Paul Goransson and Raymond Greenlaw. *Secure Roaming in 802.11 Networks*, Elsevier Science and Technical Book Group, (2007).

[12] Raymond Greenlaw, H. James Hoover, and Walter Larry Ruzzo. *Limits to Parallel Computation: P-Completeness Theory, Oxford University Press*, (1995).

[13] Sun-Yuan Hsieh and Ting-Yu Chou. Finding a Weight-constrained Maximum-density Subtree in a Tree. *Algorithms and Computation, Lecture Notes in Computer Science*, **3827**:944–953, Springer, Berlin, (2005).

[14] Robert Johnston and Clint LaFever. Hacker.mil, Marine Corps Red Team (PowerPoint Presentation). (2012).

[15] Hoong Chuin Lau, Trung Hieu Ngo, and Bao Nguyen Nguyen. Finding a Length-constrained Maximum-sum or Maximum-density Subtree and Its Application to Logistics. *Discrete Optimization*, **3(4)**:385–391, (2006).

[16] Christos H. Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*, Prentice-Hall, Inc., (1982).

[17] Shari Lawrence Pfleeger. Useful Cybersecurity Metrics. *IT Professional*, **11(3)**:38–45, (2009).

[18] Rachel Rue, Shari Lawrence Pfleeger, and David Ortiz. A Framework for Classifying and Comparing Models of Cybersecurity Investment to Support Policy and Decision-making. *Proceedings of the Workshop on the Economics of Information Security*, 23 pages, (2007).

[19] Fred B. Schneider. Blueprint for a Science of Cybersecurity, *The Next Wave*, **19(2)**:47–57, (2012).

[20] Sajjan Shiva, Sankardas Roy, and Dipankar Dasgupta. Game Theory for Cyber Security. *Proceedings of the ACM* 6^{th} *Annual Cyber Security and Information Intelligence Research Workshop*, article no. 34, April 21–23, (2010).

[21] Paul Sparrows. Cyber Crime Statistics. hackmageddon.com, retrieved October 16, 2013.

[22] Hsin-Hao Su, Chin Lung Lu, and Chuan Yi Tang. An Improved Algorithm for Finding a Length-constrained Maximum-density Subtree in a Tree. *Information Processing Letters*, **109(2)**:161–164, (2008).

[23] Jung Sung-ki. "Cyber Warfare Command to Be Launched in January." *Koreatimes.co.kr.* www.koreatimes.co.kr/www/news/nation/2013/07/205_56502.html, retrieved October 1, 2013.

[24] William Jackson. "DOD Creates Cyber Command as U.S. Strategic Command Subunit." Federal Computer Week, fcw.com, October 16, 2013.

Received 28th January 2015

Acta Cybernetica **22** (2016) 613–616.

A Note on the Emptiness of Intersection Problem for Left Szilard Languages

Erkki Mäkinen*

Abstract

As left Szilard languages form a subclass of simple deterministic languages and even a subclass of super-deterministic languages, we know that their equivalence problem is decidable. In this note we show that their emptiness of intersection problem is undecidable. The proof follows the lines of the correponding proof for simple deterministic languages, but some technical tricks are needed. This result sharpens the borderline between decidable and undecidable problems in formal language theory.

Keywords: left Szilard languages, Post Correspondence Problem, emptiness of intersection

1 Introduction

Let $G = (N, T, P, S)$ be a context-free grammar where N is the alphabet of non-terminals, T is the alphabet of terminals, P is the set of productions, and S is the start symbol. Suppose that each production in P has the form $A \to a\alpha$ where $a \in T$ and $\alpha \in N^*$. Now, if $A \to a\alpha$ and $B \to a\beta$ in P always implies $A = B$ and $\alpha = \beta$ (that is, the right hand sides start with unique terminals), we say that the grammar is a *left Szilard grammar* and the language generated is a *left Szilard language* [5]. Left Szilard languages are also known as *very simple languages* [6].

As left Szilard languages are simple deterministic languages (in the sense of Korenjak and Hoproft [4]) and super-deterministic languages (in the sense of Greibach and Friedman [1]), their equivalence problem is decidable. On the other hand "$L = L_1$?" is undecidable for a context-free language L and a left Szilard language L_1, since there are unbounded left Szilard languages, which makes the problem undecidable [3, 1].

An instance of *Post correspondence problem* (PCP) consists of two lists of words $(w_1, w_2, \ldots, w_n)$ and $(y_1, y_2, \ldots, y_n)$ over an alphabet Σ. A *solution* is a non-empty sequence of indices $i_1, \ldots, i_k$ such that $w_{i_1} \ldots w_{i_k} = y_{i_1} \ldots y_{i_k}$. It is undecidable whether such a solution exists or not for a given instance of PCP [2].

*School of Information Sciences, FI-33014 University of Tampere, Finland E-mail: {Erkki.Makinen}@uta.fi

DOI: 10.14232/actacyb.22.3.2016.4

The standard procedure to start considering undecidability problems for formal languages is to reduce PCP to the emptiness of intersection problem for context-free languages. This reduction is possible also for simple deterministic languages [4] and for super-deterministic languages [1]. This note shows that the reduction is possible also to the emptiness of intersection problem for the left Szilard languages.

2 The result

Consider an instance of PCP with lists $(w_1, w_2, \ldots, w_n)$ and $(y_1, y_2, \ldots, y_n)$ over an alphabet Σ. The text book proof (see, e.g., [2]) for the undecidability of the empiness of intersection problem for context-free languages uses grammars with productions $A \to w_1 A a_1 \mid \cdots \mid w_n A a_n \mid w_1 a_1 \mid \cdots \mid w_n a_n$ and $B \to y_1 B a_1 \mid \cdots \mid y_n B a_n \mid y_1 a_1 \mid \cdots \mid y_n a_n$, where a_i's are the unique labels of the words in the w-list and y-list. In order to transform the productions into the correct left Szilard form, we first change the places of w_i's and a_i's (resp. y_i's and a_i's), so that the unique indices can be interpreted as the unique terminals required to be in the begining of the right hand sides of the productions in a left Szilard grammar. Simultaneously, we take the mirror image of each w_i (resp. y_i) in order to keep the letters in the correct order in the resulting sentence (from right to left). Hence, if $A \to w_{i_1} \ldots w_{i_k} A a_i$ (resp. $B \to y_{i_1} \ldots y_{i_k} B a_i$) is a production in the original grammar, we change it to be $A \to a_i A w_{i_k} \ldots w_{i_1}$ (resp. $B \to a_i B \to y_{i_k} \ldots y_{i_1}$), or by using the standard notation for mirror image, we change $A \to w_i A a_i$ (resp. $B \to y_i A a_i$) to be $A \to a_i A w_i^{-1}$ (resp. $B \to a_i B y_i^{-1}$).

Both the set of A-productions and the set of B-productions constructed above contain now exactly two productions with their right hand sides starting with each of the indices a_i. The productions of the form $A \to a_i w_i$ (resp. $B \to a_i y_i$) are applied only once (as the last production) in each derivation resulting a terminal word. Therefore, we can replace each production $A \to a_i w_i$ (resp. $B \to a_i y_i$) by a production $A \to \delta_i w_i^{-1}$ (resp. $B \to \delta_i y_i^{-1}$) where δ_i's are new terminal symbols over an alphabet Δ. Notice that mirror images are needed also in these productions.

Moreover, for each symbol x in Σ, we add X, where X is a new symbol, to the set of nonterminals and the production $X \to x$ to the set of productions. Each $x \in \Sigma$ in the productions so far produced is replaced with X. All the productions are now of the required form with unique terminals in the beginning of their right hand sides.

Next we formally define the left Szilard grammars to which a given instance of PCP is reduced. Let the instance consist of the lists $W = (w_1, \ldots, w_n)$ and $Y = (y_1, \ldots, y_n)$ over Σ. Define a left Szilard grammar G_W as $(\{A\} \cup X_\Sigma, \Sigma \cup I \cup \Delta, P_W, A)$ where $X_\Sigma = \{X_{a_i} \mid a_i \in \Sigma\}$, $I = \{a_i \mid i = 1, \ldots, n\}$, $\Delta = \{\delta_i \mid i = 1, \ldots, n\}$ and P_W contains the productions $A \to a_i A w_i^{-1}$ and $A \to \delta_i w_i^{-1}$, for each w_i in the list W, and the production $X_{a_i} \to a_i$, for each $a_i \in \Sigma$. Similarly, define a left Szilard grammar G_Y as $(\{B\} \cup X_\Sigma, \Sigma \cup I \cup \Delta, P_Y, B)$ where X_Σ, I, and Δ are as in G_W, and P_Y contains the productions $B \to a_i B y_i^{-1}$ and $B \to \delta_i y_i^{-1}$, for each y_i in the list Y, and the production $X_{a_i} \to a_i$, for each $a_i \in \Sigma$.

If the PCP instance has a solution $i_1, \ldots, i_k$, we have $w_{i_1} \ldots w_{i_k} = y_{i_1} \ldots y_{i_k}$. Clearly, this happens if and only if the intersection $L(G_W) \cap L(G_Y)$ contains the word $a_{i_1} \ldots a_{i_{k-1}} \delta_{i_k} w_{i_k}^{-1} \ldots w_{i_1}^{-1} = a_{i_1} \ldots a_{i_{k-1}} \delta_{i_k} y_{i_k}^{-1} \ldots y_{i_1}^{-1}$.

We have proved the following theorem.

Theorem 1. *The emptiness of intersection problem is undecidable for left Szilard languages.*

We end this chapter by an example of the above construction. Let the lists $W = (a, abaaa, ab)$ and $Y = (aaa, ab, b)$ form an instance of PCP. The words in the lists contain letters a and b; hence, we have $\Sigma = \{a, b\}$. The sequence of indices $2-1-1-3$ is a solution for this instance and the common string corresponding to these indices is aba^6b. The corresponding left Szilard grammar G_W has the productions $A \to 1AX_a$, $A \to 1_\delta X_a$, $A \to 2AX_aX_aX_aX_bX_a$, $A \to 2_\delta X_aX_aX_aX_bX_a$, $A \to 3AX_bX_a$, $A \to 3_\delta X_bX_a$, $X_a \to a$, and $X_b \to b$. Similarly, the left Szilard grammar G_Y has the productions $B \to 1BX_aX_aX_a$, $B \to 1_\delta X_aX_aX_a$, $B \to 2BX_bX_a$, $B \to 2_\delta X_bX_a$, $B \to 3BX_b$, $B \to 3_\delta X_b$, $X_a \to a$, and $X_b \to b$.

The word corresponding to $2-1-1-3$ can be generated in G_W and G_Y as follows:

$$A \Rightarrow 2AX_a^3X_bX_a \Rightarrow 21AX_a^4X_bX_a \Rightarrow 211AX_a^5X_bX_a$$
$$\Rightarrow 2113_\delta X_bX_a^6X_bbX_a \Rightarrow^+ 2113_\delta ba^6ba$$

and

$$B \Rightarrow 2BX_bX_a \Rightarrow 21BX_a^3X_bX_a \Rightarrow 211BX_a^6X_bX_a$$
$$\Rightarrow 2113_\delta X_bX_a^6X_bX_a \Rightarrow^+ 2113_\delta ba^6ba.$$

3 Discussion

The emptiness of intersection problem for context-free languages is the basic undecidable problem in formal language theory, as in most treatments it transmits the undecidability of Turing machine computations to language theory. A natural question then is to find the simplest class of languages for which this transmission is possible. Previously, the classes of simple deterministic languages and superdeterministic languages have been known to be enough for the reduction. This note shows that the structure of PCP can be presented even in the terms of left Szilard languages.

References

[1] Greibach, S.A., and Friedman, E.P., Superdeterministic PDAs: A subcase with a decidable inclusion problem. *Journal of the ACM* 27(4):675–700, 1980.

[2] Hopcroft, J.E., Motwani, R., and Ullman, J.D. *Introduction to Automata Theory, Languages, and Computation, 2nd Edition.* Addison-Wesley, 2001.

[3] Hunt III, H.B., and Rangel, J.L. Decidability of equivalence, containment, intersection, and separability of context-free languages. In *Proc. 16th Annual Symposium on Foundations of Computer Science*, pages 144-149, 1975.

[4] Korenjak, A.J., and Hopcroft, J.E. Simple deterministic languages. In *IEEE Conference Record of Seventh Annual Symposium on Switching and Automata Theory*, pages 36-46, 1966.

[5] Mäkinen, E. On context-free derivations. *Acta Universitatis Tamperensis* Ser. A, Vol. 198, 1985.

[6] Yokomori, T. Polynomial-time identification of very simple grammars from positive data. *Theoretical Computer Science* 298,1:179-206, 2003.

Received 19th August 2015

Acta Cybernetica **22** (2016) 617–631.

Performance Modeling of Finite-Source Cognitive Radio Networks

Béla Almási*, Tamás Bérczes*, Attila Kuki*, János Sztrik* and Jinting Wang†

Abstract

This paper deals with performance modeling aspects of radio frequency licensing. The utilization of mobile cellular networks can be increased by the idea of the cognitive radio. Licensed users (Primary Users - PUs) and normal users (Secondary Users - SUs) are considered. The main idea is, that the SUs are able to access to the available non-licensed radio frequencies.

A finite-source retrial queueing model with two non independent frequency bands (considered as service units) is proposed for the performance evaluation of the system. A service unit with a priority queue and another service unit with an orbit are assigned to the PUs and SUs, respectively. The users are classified into two classes: the PUs have got a licensed frequency, while the SUs have got a frequency band, too but it suffers from the overloading. We assume that during the service of the non-overloaded band the PUs have preemptive priority over SUs. The involved inter-event times are supposed to be independent and exponentially distributed random variables.

The novelty of this work lies in the fact that we consider the effect of retrial phenomenon of SUs in performance modeling of radio frequency licensing by using a finite-source queueing model which takes the unreliability of radio transmission into account for the first time. In the literature, most work studied the performance of cognitive radio networks under a mixed spectrum environment of licensed and unlicensed bands where the blocked SUs and the preempted SUs are forced to leave the system forever when there are no idle channels in the system. But in practical situation, the blocked SUs and the preempted SUs may do not leave the system forever and try to continue their services after random amount of time.

By the help of an appropriate continuous time Markov chain using MOSEL (MOdeling Specification and Evaluation Language) tool several numerical examples are provided showing the effects of different input parameters on the main performance measures of the cognitive radio networks.

Our primary focus is to determine an optimal number of SUs, where at the

*University of Debrecen, Hungary, E-mail: almasi.bela,berczes.tamas, kuki.attila, sztrik.janos@inf.unideb.hu

†Beijing Jiaotong University, China E-mail: jtwang@bjtu.cn

DOI: 10.14232/actacyb.22.3.2016.5

secondary band the gained utilization, that is when switching to the cognitive radio, has a maximum value.

Keywords: cognitive radio networks, performance evaluation, finite-source retrial queueing systems, modeling tools

1 Introduction

The increasing demand for mobile communication has produced a bottleneck point in the radio frequency licensing. In order to increase the performance of mobile cellular networks the idea of the cognitive radio was introduced (see [15]). The users having "cognitive" wireless devices (called Secondary Users, SUs or Normal Users) are able to access to the available non-licensed radio frequencies. This access may not affect or disturb the communication of the licensed users (called Primary Users, PUs), i.e. the "cognitive" radio devices must intelligently release the unlicensed spectrum if a licensed user appears.

The teletraffic theory and modeling the attempt of repeated calls have been investigated widely in the past decades. Interesting result can be found on models, measurements, blocking probabilities, and so on, e.g. in [10], [13], [14], and [21]. The cognitive radio networks (CRN) have been a hot research area recently. As we could realize queueing theory can be successfully applied to establish mathematical models for variety of telecommunication systems (see e.g. [1], [2], [3], [6], [7], [8], [11], [12], [16], [17], [18],). These models can be used to calculate performance measures like mean delay, mean waiting time, utilization of the frequency bands etc. In many cases infinite preemptive priority queueing models are applied (see for example [5],[18]) for modeling and analysis. As the cognitive radio environments are used also in small sized radio network cells (picocells or femtocells) finite- source queueing models can be also appropriate for modeling. Recently in [19] Wong and Foh has introduced a finite-source queueing model to study the performance of a CRN with one frequency band (i.e. one queueing service station). This paper gave us an inspiration to develop a more realistic model.

In the present paper we introduce a finite-source queueing model with two (non independent) frequency bands (channel, server). As it is widely used in CRN modeling (see e.g. [8]) the users are classified into two classes: the Primary Users (PUs) have got a licensed frequency, which does not suffer from overloading feature. The Secondary Users (SUs, or Normal Users) have got a frequency band too, but it suffers from the overloading. If the band of the SUs is engaged then a newly arriving SU request may use the band of the PUs (which is not licensed for the SUs) in a cognitive way: the non-licensed frequency must be released by the SU if a PU request appears. In our environment the band of the PUs is modeled by a queue, where the requests from the PUs' class has preemptive priority over the SUs' request: the SU's request is subject to an immediate "handover" from the PU's band to the SU's band if a PU's request appears. The band of the SUs is described by a retrial queue: if the band is free when the request arrives then it is

transmitted. Otherwise the request goes to the Orbit if both bands are busy. The transmission of the requests from the orbit will be retried after a random holding time.We assume that the radio transmission is not reliable, that is the transmission will fail with a non-zero probability p for both channels. If this happens then the request retransmission process will start immediately.

Hence it should be noted that the novelty of our work lies in the fact that we consider the effect of retrial phenomenon of SUs in performance modeling of radio frequency licensing by using a finite-source queueing model which takes the unreliability of radio transmission into account for the first time. In the literature, most work studied the performance of cognitive radio networks under a mixed spectrum environment of licensed and unlicensed bands where the blocked SUs and the preempted SUs are forced to leave the system forever when there are no idle channels in the system. But in practical situation, the blocked SUs and the preempted SUs may do not leave the system forever and try to continue their services after random amount of time. This is a motivation of our research and model improvement.

The rest of the paper is organized as follows. Section 2 describes the precise mathematical model when a multi-dimensional continuous time Markov-chain is defined for describing the system's dynamics. Formulas of the most important performance measures are also discussed here. For presentation of numerical results the MOdeling Specification and Evaluation Language (MOSEL see [4]) tool is used. In Section 3 the most important system characteristics (utilizations, response times) are determined in order to verify the model. Series of experience are carried out to study how the number of the SUs' influences the average gain of the cognitive radio related to the non-cognitive one. At the end of this section some empirical optimization results are shown. Finally, the paper ends with conclusions.

2 System Model

A finite source queueing system, as illustrated in Fig.1 is used to model the considered cognitive radio network. The queueing system contains two interconnected, not independent sub-systems. The first part is for the requests of the PUs. The number of sources is denoted by N_1. These sources generate high priority requests with exponentially distributed inter-request times with parameter λ_1. The generated requests (jobs, packets) are sent to a single server unit (Primary Channel Service - PCS) with a preemptive priority queue. The service times are exponentially distributed with parameter μ_1. The second part is a finite-source retrial queueing system. The requests from the SUs are generated here. There are N_2 sources, the inter-request times are assumed to be exponentially distributed random variables with parameter λ_2. The single server unit (Secondary Chanel Service - SCS) works according to exponentially distributed service times with parameter μ_2.

A generated high priority packet goes to the primary service unit (i.e. to the radio band licensed by the PUs). If the unit is idle, the service of the packet begins immediately. If the server (i.e. radio channel) is engaged with a high priority

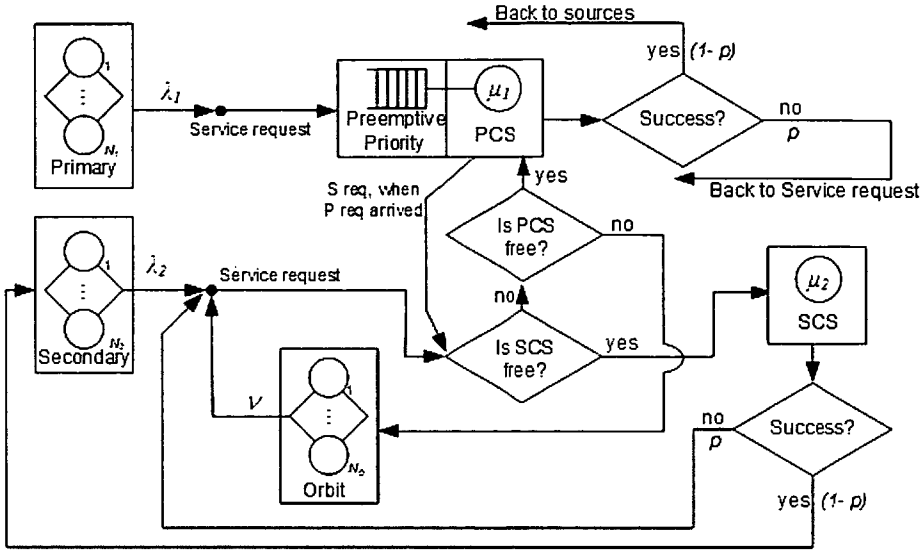


Figure 1: A priority and a retrial queue with components

request, the packet joins to the preemptive priority queue. When the unit is engaged with a request from SUs, the service is interrupted and the interrupted low priority task is sent back to the SCS. Depending on state of the secondary channel the interrupted job is directed to either the server or the orbit. At the PCS the service of the incoming request (which has interrupted the service of the task from SUs) begins. The transmission through the radio channel may produce errors, which can be discovered after the transmission (i.e. after the service). In the model this case has a probability p, and the failed packet is sent back to the appropriate service unit (high priority request to the PCS, low priority request to the SCS). When the submission is successful (with probability $1-p$), the request goes back to the source (to the PUs or SUs, depending on the packet, respectively).

A generated request from SUs looks for the secondary service unit. If the SCS is idle, the service of the task begins immediately. If the SCS is busy, the packet checks the primary service unit. In case of an idle PCS, the service of the low priority packet begins at the high priority channel (PCS). If the PCS is busy, the packet goes to the orbit. From the orbit it retries to be served after an exponentially distributed time with parameter ν. After service, the same transmission failure with the same probability can occur as in the PCS segment. Here the failed request is sent back to the SCS, the token of the successfully submitted packet goes back to the secondary source.

This functionality of the system can be seen on Fig.1.

To create a stochastic process describing the behavior of the system the following notations are introduced

- $k_1(t)$ is the number of high priority sources at time t,
- $k_2(t)$ is the number of low priority (normal) sources at time t,
- $q(t)$ denotes the number of high priority requests in the priority queue at time t,
- $o(t)$ is the number of requests in the orbit at time t.
- $y(t) = 0$ if there is no job in the PCS unit, $y(t) = 1$ if the PCS unit is busy with a job coming from the high priority class, $y(t) = 2$ when the PCS unit is servicing a job coming from the secondary class at time t
- $c(t) = 0$ when the SCS unit is idle and $c(t) = 1$, when the SCS is busy at time t.

It is easy to see that

$$k_1(t) = \begin{cases} N_1 - q(t), & y(t) = 0, 2 \\ N_1 - q(t) - 1, & y(t) = 1 \end{cases}$$

and

$$k_2(t) = \begin{cases} N_2 - o(t) - c(t), & y(t) = 0, 1 \\ N_2 - o(t) - c(t) - 1, & y(t) = 2. \end{cases}$$

The network input parameters are collected in Table 1.

Table 1: List of network parameters

Parameter	Maximum	Value at t
Active primary sources	N_1	$k_1(t)$
Active secondary sources	N_2	$k_2(t)$
Primary generation rate		λ_1
Secondary generation rate		λ_2
Total gen. rate	$\lambda_1 N_1 + \lambda_2 N_2$	$\lambda_1 k_1(t) + \lambda_2 k_2(t)$
Requests in priority queue	$N_1 - 1$	$q(t)$
Requests in orbit	$N_2 - 1$ (orbit size)	$o(t)$
Primary service rate		μ_1
Secondary service rate		μ_2
Retrial rate		ν
Error rate		p

To obtain the steady-state probabilities and performance measures within the Markovian framework, the mathematical tractability of the proposed model should be preserved. Therefore, we follow the classical approach frequently applied in the theory of retrial queues for the performance evaluation of wireless cellular networks, namely, the distributions of inter-event times (i.e., request generation times for low and high priority jobs, service time, retrial time) presented in the system are assumed to be exponentially distributed and totally independent of each other.

Hence the state of the network at a time t can be described by a Continuous Time Markov Chain (CTMC) with 4 dimensions:

$$X(t) = (y(t), q(t); c(t), o(t))$$

The steady-state distributions are denoted by

$$P(y, q, c, o) = \lim_{t \to \infty} P(y(t) = y, q(t) = q, c(t) = c, o(t) = o)$$

Note that in the present case, the unique stationary distribution always exists, because the underlying CTMC is irreducible and the state space of the CTMC is finite. The computation of this distribution is described e. g. in [9]. For computing the steady-state probabilities and the system characteristics, the MOSEL-2 software tool is used. These computations are similar to the ones described in, for example [20].

As soon as we have calculated the distributions defined above, the most important steady-state system performance measures can be obtained in the following way

- *Utilization of the primary server with respect to primary users*

$$U_{11} = \sum_{q=0}^{N_1-1} \sum_{c=0}^{1} \sum_{o=0}^{N_2-c} P(1, q, c, o)$$

- *Utilization of the primary server with respect to secondary users*

$$U_{12} = \sum_{q=0}^{N_1-1} \sum_{c=0}^{1} \sum_{o=0}^{N_2-c} P(2, q, c, o)$$

- *Utilization of the primary server*

$$U_1 = U_{11} + U_{12}$$

- *Utilization of the secondary server*

$$U_2 = \sum_{y=0}^{2} \sum_{q=0}^{N_1-1} \sum_{o=0}^{N_2-1} P(y, q, 1, o)$$

- *Average number of jobs in queue*

$$\overline{Q} = \sum_{y=0}^{2} \sum_{q=0}^{N_2-1} \sum_{c=0}^{1} \sum_{o=0}^{N_2-c} qP(y, q, c, o)$$

- *Average number of jobs in the orbit*

$$\overline{O} = \sum_{y=0}^{2} \sum_{q=0}^{N_2-1} \sum_{c=0}^{1} \sum_{o=0}^{N_2-c} oP(y,q,c,o)$$

- *Average number of jobs of primary users in the network*

$$\overline{M_1} = \overline{Q} + U_{11}$$

- *Average number of jobs of secondary users in the network*

$$\overline{M_2} = \overline{O} + U_{12} + U_2$$

- *Average number of jobs in the network*

$$\overline{M} = \overline{M_1} + \overline{M_2}$$

- *Average number of active primary users*

$$\overline{\Lambda_1} = N_1 - \overline{M_1}$$

- *Average number of active secondary users*

$$\overline{\Lambda_2} = N_2 - \overline{M_2}$$

- *Average generation rate of primary users*

$$\overline{\lambda_1} = \lambda_1 \overline{\Lambda_1}$$

- *Average generation rate of secondary users*

$$\overline{\lambda_2} = \lambda_2 \overline{\Lambda_2}$$

- *Mean response time of primary user's jobs*

$$\overline{T_1} = \frac{\overline{M_1}}{\overline{\lambda_1}}$$

- *Mean response time of secondary user's jobs*

$$\overline{T_2} = \frac{\overline{M_2}}{\overline{\lambda_2}}$$

Table 2: Numerical values of model parameters

Case studies								
No.	N_1	N_2	λ_1	λ_2	μ_1	μ_2	ν	p
Fig. 2	10, 20	50	$x-axis$	0.03	1	1	20	0.1
Fig. 3	10	$x-axis$	0.02	0.03	1	1	20	0.1
Fig. 4	10	10	0.02, 0.08	0.03, 0.08	1	1	$x-axis$	0.1
Fig. 5	10, 20	50	$x-axis$	0.03	1	1	20	0.1
Fig. 6	10	$x-axis$	0.02	0.03	1	1	20	0.1
Fig. 7	10	$x-axis$	0.02	0.03	1	1	20	0.1
Fig. 8	10	$x-axis$	0.02	0.03	1	1	20	0.1
Fig. 9	10	$x-axis$	0.02	0.03	1	1	20	0.1

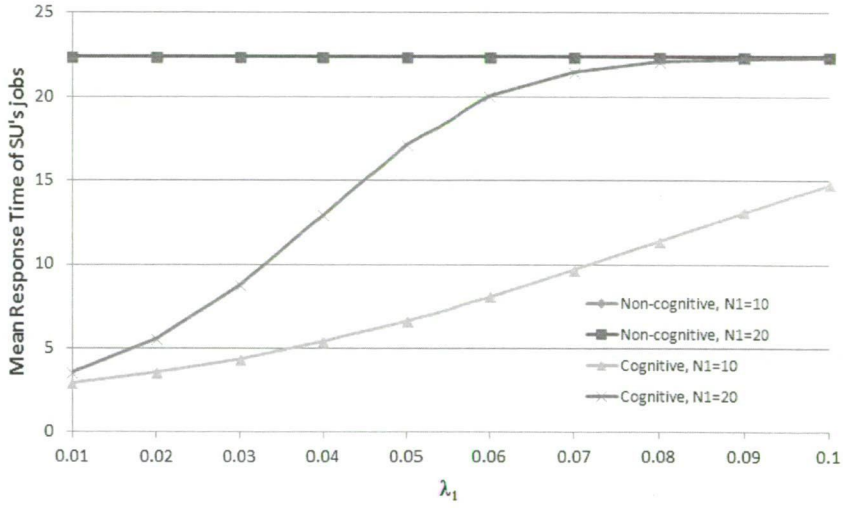


Figure 2: Mean Response Time of SU's jobs vs. λ_1

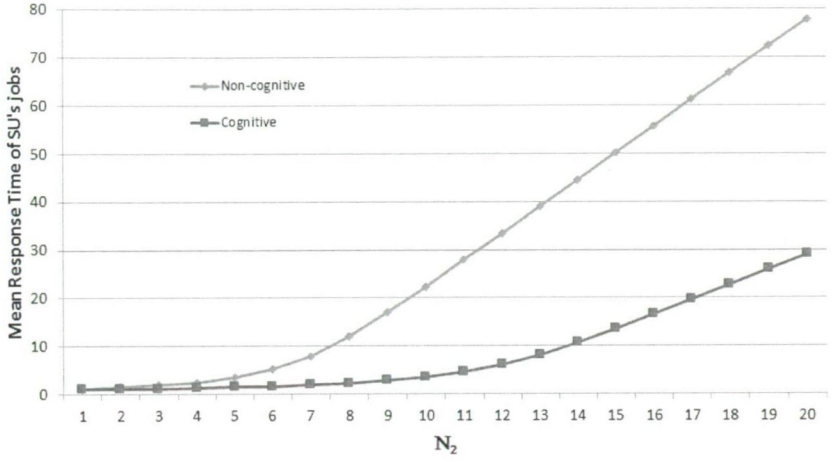


Figure 3: Mean Response Time of SU's jobs vs. N_2

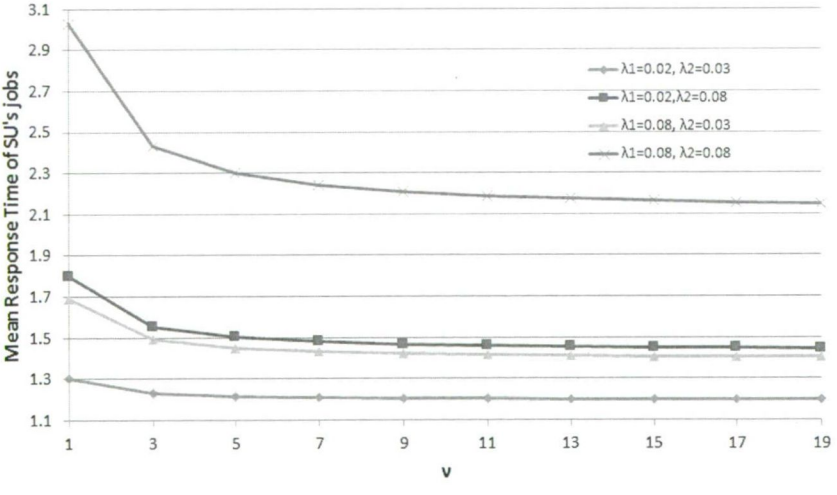


Figure 4: Mean Response Time of SU's jobs vs. ν

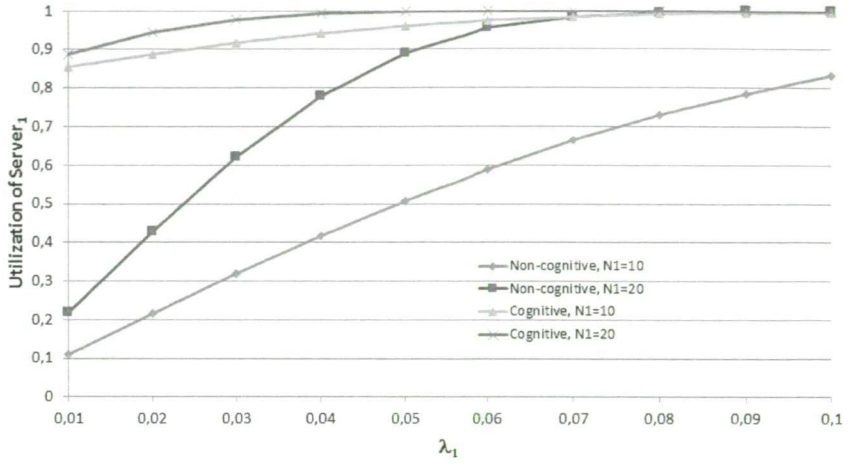


Figure 5: Utilization of PCS vs. λ_1

3 Numerical results

Investigating the functionality and the behavior of the system several numerical calculations were performed. From the steady-state probabilities computed by MOSEL-2 tool the most interesting performance characteristics were obtained, which are graphically presented in this section. The numerical values of model parameters are collected in Table 2.

In the calculations two operational modes were applied. The Non-Cognitive Radio Mode (Non-cognitive on figures), where the secondary users are not enabled to use the high priority primary channel, and the Cognitive Radio Mode (Cognitive on figures), where the secondary users can use the idle primary channel.

On Figure 2 the mean response times of the secondary jobs are displayed as a function of the primary generation rate. Beside the cognitive and non-cognitive radio feature, two different values of the number of primary sources are introduced. In the non-cognitive cases the two subsystems (the flow of primary and secondary jobs) are independent, thus the value of λ_1 and the number of primary source have no effect for this system characteristic. The two non-cognitive lines are identical. In the cognitive case it can be seen, that increasing the value of λ_1, the secondary jobs have less opportunities to use the primary channel, the response times significantly increase.

Figure 3 shows the mean response times of the secondary jobs in cases of different numbers of the secondary source. Natural way, the more secondary users try to use the system, the greater the response time is. Comparing the cognitive and the non-cognitive results, the benefit (gain) of the cognitive radio network is displayed. For low numbers of N_2 the response times are almost the same, but increasing the number of the secondary source, the difference between the response times

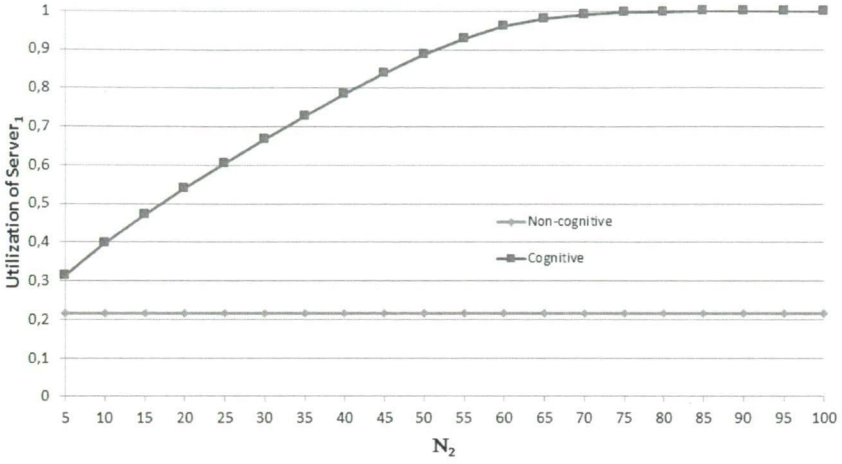


Figure 6: Utilization of PCS vs. N_2

increases, as well.

On Figure 4 the effect of the time spent in orbit (it was modeled by a variable retrial rate) to the response time of SCS is displayed.

On Figure 5 the utilization of the PCS is displayed as a function of λ_1. For small values of the primary generation rate the two cognitive lines are highly above the non-cognitive ones, because in cognitive case the idle periods of the PCS are filled up with jobs from the secondary source.

The next figure (Figure 6 the utilization of PCS is shown as a function of the numbers of secondary source. In non-cognitive case the number of N_2 has no effect for this characteristic. The jobs from primary and secondary sources work independently. The utilization is constant. In cognitive case the increasing number of the secondary source increases the utilization of PCS, more and more jobs from the secondary source try to use the PCS.

On Figure 7 the same investigation is performed for the SCS. In non-cognitive case the server load reaches the maximum utilization soon, because the jobs are not able to use the PCS. In cognitive case the PCS also can serve secondary jobs, thus the utilization of SCS will reach the value of 1 only at higher numbers of secondary source.

Figure 8 shows the relative difference (the difference is divided by N_2) of mean orbit size in non-cognitive and cognitive cases. This is the increment of orbit size for one user in function of number of secondary users. When the size of the orbit fills up, this increment has a local maximum value. After this point, the increment will be decreasing, because the orbit still remains full and the number of users are increasing.

On the last figure (Figure 9) the difference of utilizations of SCS (in non-cognitive and cognitive cases) is displayed. With these parameter setup the optimal

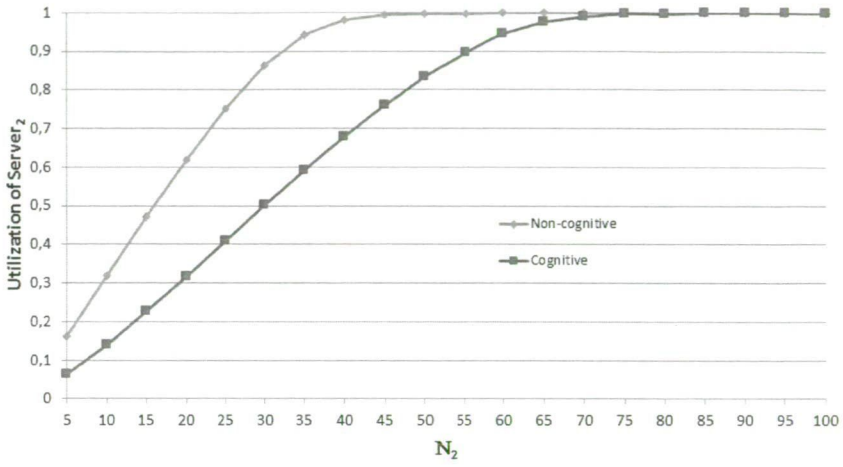


Figure 7: Utilization of SCS vs. N_2

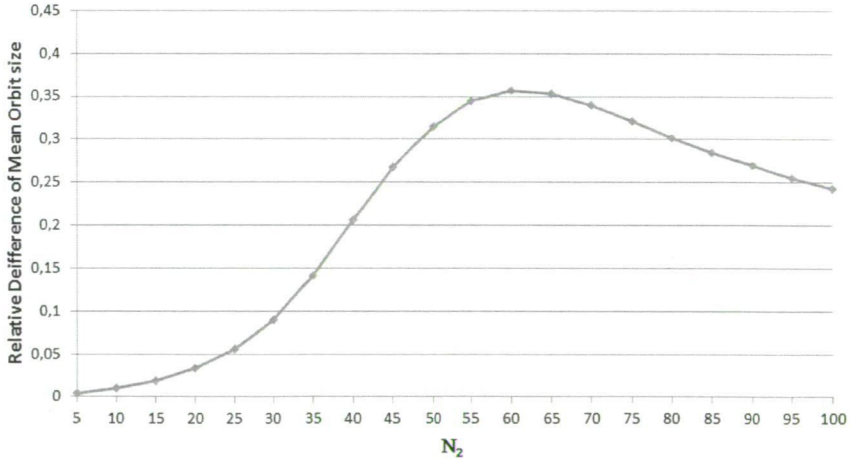


Figure 8: Relative Difference of Mean Orbit Size vs. N_2

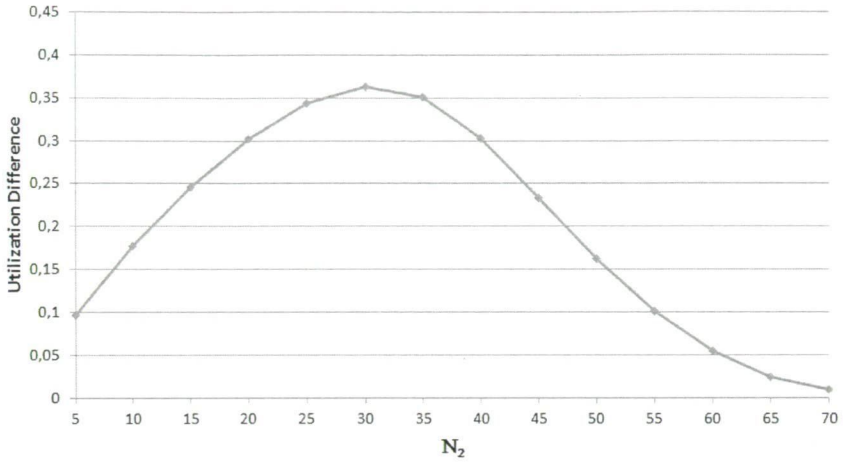


Figure 9: Difference of utilization of SCS vs. N_2

point of utilization gains can be found at 30, i.e. this is the number of secondary users, when it is worth to change to the cognitive radio devices.

4 Conclusions and further questions

In this paper a novel finite-source retrial queueing model was proposed with two bands servicing primary and secondary users in a cognitive radio network. Primary users have preemptive priority over the secondary ones in servicing at primary channel. At the secondary channel an orbit is installed for the secondary jobs finding the server busy upon arrival. The MOSEL tool was used to carry numerical calculations illustrating the effect of some input parameters on the main performance measures. The primary focus was to determine an optimal number of secondary users, where at the secondary band the gained utilization, that is when switching to the cognitive radio, has a maximum value.

Interesting further works could be to investigate the case of more than one frequency chanels. Other interesting problem is, when the users are able to switch their classes, i.e. a secondary user can advance into the class of primary users. Finally, other distributions, e.g. hipo-exponential distribution can be investigated, as well.

5 Acknowledgments

The publication was supported by the TÁMOP 4.2.2. C-11/1/KONV-2012-0001 project. The project has been supported by the European Union, co-financed by the European Social Fund.

The present work was granted by Chinese-Hungarian Bilateral Cooperation in Science and Technology project TéT 12 CN-1-2012-0009 (Chinese Grant No. 2013-83-6-34).
The research work of Jinting Wang was partially supported by the National Science Foundation of China (Grant Nos. 11171019 and 71390334), and the Program for New Century Excellent Talents in University (No. NCET-11-0568).

References

[1] Artalejo, J.R. Accessible bibliography on retrial queues. *Mathematical and Computer Modelling*, 30:1–6, 1999.

[2] Artalejo, J.R. and Gomez-Corral, A. *Retrial Queueing Systems.* Springer, Berlin, 2008.

[3] Basharin, G. P., Samouylov, K. E., Yarkina, N. V., and Gudkova, I. A. A new stage in mathematical teletraffic theory. *Automation Remote Control*, 70(12):1954–1964, 2009.

[4] Begain, K., Bolch, G., and Herold, H. *Practical Performance Modeling, Application of the MOSEL Language,*. Kluwer Academic Publisher, Boston, 2001.

[5] Devroye, N., Vu, M., and Tarock, V. Cognitive radio networks. *IEEE Signal Processing Magazine*, 25:12–23, 2008.

[6] Do, T. Van, Chakka, R., and Sztrik, J. Spectral expansion solution methodology for QBD-M processes and applications in Future Internet engineering. *Advanced Computational Methods for Knowledge Engineering, Studies in Computational Intelligence*, 479:131–142, 2013.

[7] Feng, W. and Kowada, M. Performance analysis of wireless mobile networks with queueing priority and guard channels. *International Transactions on Operational Research*, 15:481–508, 2008.

[8] Gao, S. and Wang, J. Performance analysis of a cognitive radio network based on a preemptive priority retrial queue. *International Journal of Computer Mathematics*, GCOM-2013-0621-A, 2014.

[9] Gross, D.; Shortie, J. F.; Thompson J. M. & Harris C. M. *Fundamentals of Queueing Theory, Fourth Edition.* John Wiley & Sons, Inc., 2008.

[10] Jonin, G. L. The systems with repeated calls: Models, measurements, results. *Fundamentals of Teletraffic Theory. Proceedings of the Third International Seminar on Teletraffic Theory*, pages 197–208, 1980.

[11] Lakatos, L., Szeidl, L., and Telek, M. *Introduction to queuing systems with telecommunication applications.* Springer, Heidelgerg, 2013.

[12] Lee, Y., Park, C. G., and Sim, D. B. Cognitive radio spectrum access with prioritized secondary users. *Applied Mathematics & Information Sciences*, 6:595–601, 2012.

[13] Marsan, M. A.; De Carolis, G.; Leonardi E.; Lo Cigno R. & Meo M. An approximate model for computation of blocking probabilities in cellular networks with repeated calls. *Telecommunication Systems*, 15:53–62, 2000.

[14] Marsan, M. A.; De Carolis, G.; Leonardi E.; Lo Cigno R. & Meo M. Efficient estimation of call blocking probabilities in cellular mobile telephony networks with customer retrials. *IEEE Journal on Selected Areas in Communications*, 19:332–346, 2001.

[15] Mitola, J. *Cognitive radio: an integrated agent architecture for software defined radio.* PhD thesis, KTH, 2000.

[16] Ponomarenko, L., Kim, C. S., and Melikov, A. *Performance analysis and optimization of multi-traffic on communication networks.* Springer, Heidelberg, 2010.

[17] Stasiak, M., Glabowski, M., Wishniewski, A., and Zwierzykowski, P. *Modeling and dimensioning of mobile networks.* John Wiley, Chichester, 2011.

[18] Wang, L.C., Wang, C.W., and Adachi, F. Load-balancing spectrum decision for cognitive radio networks. *IEEE Journal on Selected Areas of Communications*, 29:757–769, 2011.

[19] Wong, E. W. M. and Foh, C. H. Analysis of cognitive radio spectrum access with finite user population. *IEEE Communications Letters*, 13, 2009.

[20] Wüchner, P., Sztrik, J., and de Meer, H. Modeling wireless sensor networks using finite-source retrial queues with unreliable orbit. *Springer Lecture Notes in Computer Science*, 6821:275–285, 2011.

[21] Zukerman, M. & Chia Min Lee. Performance bounds for cellular mobile communications networks with repeated attempts. *Vehicular Technology Conference, 2001. VTC 2001 Spring. IEEE VTS 53rd*, 2:996–1000, 2001.

Received 25th April 2014

Acta Cybernetica **22** (2016) 633–655.

One-Pass Reductions*

Sándor Vágvölgyi†

Abstract

We study OI and IO one-pass reduction sequences with term rewrite systems. We present second order decidability and undecidability results on recognizable tree languages and one-pass reductions. For left-linear TRSs, the second order OI inclusion problem and the second order OI reachability problem are decidable, the second order OI joinability problem is undecidable. For right-linear TRSs, the second order common IO ancestor problem is undecidable.

Keywords: term rewrite systems, OI and IO one-pass reductions, tree automata

1 Introduction

A term rewrite system (TRS for short) R reduces a term in a nondeterministic way along which it does many choices. Traditional term rewriting is the exhaustive application of R to a term until no more rules apply. However, this procedure is usually not adequate for most applications, for example for program transformation. Because, in general, there is no bound for the lengths of the possible reduction sequences, or R is not confluent.

To overcome the above problems, researchers implemented and studied various types of traversal reductions. Program transformation operates on the syntax tree of a program applying rewrite rules along a traversal of the tree: it visits all tree nodes in a certain visiting order and applies a rewrite rule at each node at most once [2, 3, 14, 15]. They distinguish between the standard visiting orders top-down (order: root, subtrees) and bottom-up (order: subtrees, root). Dauchet and De Comité [4], Seynhaeve et al. [12] studied outside-in (OI) and inside-out (IO) one-pass reductions, which are different from the above top-down and bottom-up traversals, respectively, in that all reduction steps can be carried out mainly independently from each other. During a reduction step, the left-hand side of an

*The publication is supported by the European Union and co-funded by the European Social Fund. Project title: "Telemedicine-focused research activities on the field of Mathematics, Informatics and Medical sciences" Project number: TÁMOP-4.2.2.A-11/1/KONV-2012-0073

†Department of Foundations of Computer Science, University of Szeged, Árpád tér 2, H-6720 Szeged, Hungary E-mail: vagvolgy@inf.u-szeged.hu

DOI: 10.14232/actacyb.22.3.2016.6

applied rule does not overlap with the already rewritten parts of the term, only the values of the substituted subterms depend on the order of the reduction steps. One may proceed in two ways. Along an OI one-pass reduction sequence, we proceed from the root to the leaves. Along an IO one-pass reduction sequence, we proceed from the leaves to the root. Fülöp et al. [5] studied two other very restrictive strategies of term rewriting: and one-pass root-started rewriting and one-pass leaf-started rewriting. They differ from the OI and IO one-pass reductions, respectively, in that the rewriting always concern positions immediately adjacent to the already rewritten parts of the term. Consequently, they establish a much more restricted way of computing.

Reachability is a fundamental problem that appears in several areas of computer science: finite- and infinite-state concurrent systems, computational models like cellular automata and Petri nets, program analysis, discrete and continuous systems, time critical systems, hybrid systems, TRSs, etc. [13]. For TRSs, reachability problem is the following: given a TRS R, and two terms s and t, decide whether s can be rewritten into t with a finite number of rewriting steps of R. Ground reachability problem is the restriction of the reachability problem to ground terms. Ground reachability and unreachability proofs can be used as general purpose verification techniques for the systems modeled by rewriting [9].

Gilleron and Tison [10] introduced and studied the second order reachability problem and the second order sentential form inclusion problem for TRSs. They [10] asked whether the set of sentential forms of the trees of a recognizable tree language overlaps with a given recognizable tree language, and whether the set of sentential forms of the trees of a recognizable tree language is a subset of a given recognizable tree language, respectively. Observe that they [10] defined a second order decidability problem from a first order one by substituting recognizable tree languages for terms. Along this line of research, Fülöp et al. [5] introduced and studied second order decidability problems: the one-pass root-started sentential form inclusion problem, and the one pass leaf-started sentential form inclusion problem. Moreover, Seynhaeve et al. [12] presented and studied the second order IO inclusion problem.

In the light of the above problems, we study the following eight second order decidability problems. Some of them, for instance the second order OI common ancestor problem, are introduced in this paper following the above research line. The terms appearing in an OI (resp. IO) one pass reduction sequence are called the OI (resp. IO) sentential forms of the initial term. For a tree language L, the set of all OI (resp. IO) sentential forms of the elements of L is denoted by $SFOI(L)$ (resp. $SFIO(L)$). First we present the problems concerning OI one pass reducing.

Second order OI inclusion problem.
Instance: A TRS R and recognizable tree languages L and M over Σ.
Question: Is $SFOI(L) \subseteq M$?

Second order OI reachability problem.
Instance: A TRS R and recognizable tree languages L and M over Σ.
Question: Is $SFOI(L) \cap M \neq \emptyset$?

Second order OI joinability problem.
Instance: A TRS R and recognizable tree languages L and M over Σ.
Question: Is $SFOI(L) \cap SFOI(M) \neq \emptyset$?

Second order OI common ancestor problem.
Instance: A TRS R and recognizable tree languages L and M over Σ.
Question: Is there a term $t \in T_\Sigma(X)$ such that $SFOI(t) \cap L \neq \emptyset$ and $SFOI(t) \cap M \neq \emptyset$?

We define the IO counterparts of the above problems replacing OI by IO.

Seynhaeve et al. showed that for left-linear TRSs and right-linear TRSs, the second order IO inclusion problem is decidable, see Proposition 4 in [12]. Fülöp et al. [5] showed that for left-linear TRSs, the one-pass root-started sentential form inclusion problem, the counterpart of the second order IO inclusion problem, and the one pass leaf-started sentential form inclusion problem, the counterpart of the second order OI inclusion problem, are decidable. Seynhaeve et al. showed that for right-linear TRSs, the one-pass root-started sentential form inclusion problem is decidable, see Proposition 5 in [12].

In Section 2, we present our notations and basic definitions. Then we show the following. For left-linear TRSs, the second order OI inclusion problem and the second order OI reachability problem are decidable, see Section 3. For left-linear TRSs, the second order OI joinability problem is undecidable, see Section 4. For right-linear TRSs, the second order common IO ancestor problem is undecidable, see Section 5. In Section 6, we present our concluding remarks and open problems. We sum up the existing results in the literature and our contribution in Table 1, where OI, IO, ll, rl, and ances. abbreviate one-pass OI reduction, one-pass IO reduction, left-linear, right-linear, and ancestor, respectively. Each question mark signifies an open problem.

decidability of	inclusion	reachability	joinability	common ances.
OI	decidable for ll Theorem 1	decidable for ll Theorem 1	undecidable for ll Theorem 2	?
IO	decidable for ll and for rl [12], Prop. 4	?	?	undecidable for rl Theorem 3

Table 1: Summary of results

2 Preliminaries

We recall and invent some notations, basic definitions and terminology which will be used in the rest of the paper. Nevertheless the reader is assumed to be familiar with the basic concepts of term rewrite systems and of tree language theory [1, 7, 8].

2.1 Terms

The set of nonnegative integers is denoted by N, and N^* stands for the free monoid generated by N with empty word λ as identity element. For a word $\alpha \in N^*$, $length(\alpha)$ stands for the length of α. Consider the words $\alpha, \beta, \gamma \in N^*$ such that $\alpha = \beta\gamma$. Then we say that β is a prefix of α, and that α is an extension of β; and we write $\beta \preceq \alpha$. If $\gamma \neq \lambda$, then β is a proper prefix of α, and we write $\beta \prec \alpha$.

A ranked alphabet is a finite set Σ in which every symbol has a unique rank in N. For $m \geq 0$, Σ_m denotes the set of all elements of Σ which have rank m. The elements of Σ_0 are called constants. Throughout the paper we assume that $\Sigma_0 \neq \emptyset$. That is, we have at least one constant in Σ.

For a set of variables Y and a ranked alphabet Σ, $T_\Sigma(Y)$ denotes the set of Σ-terms (or Σ-trees) over Y. $T_\Sigma(\emptyset)$ is written as T_Σ. A term $t \in T_\Sigma$ is called a ground term. A tree $t \in T_\Sigma(Y)$ is linear if any variable of Y occurs at most once in t. We specify a countable set $X = \{ x_1, x_2, \dots \}$ of variables which will be kept fixed in this paper. Moreover, we put $X_m = \{ x_1, \dots, x_m \}$, for $m \geq 0$. Hence $X_0 = \emptyset$.

For a term $t \in T_\Sigma(X)$, the height $height(t)$ and the yield $yd(t)$ and the set of positions $pos(t) \subseteq N^*$ of t are defined by tree induction.

- If $t \in \Sigma_0 \cup X$, then $height(t) = 0$, $yd(t) =$(if $t \in \Sigma_0$ then λ else t), and $pos(t) = \{ \lambda \}$.
- If $t = f(t_1, \dots, t_m)$ with $f \in \Sigma_m$, $m > 0$, then
  $height(t) = 1 + \max\{height(t_i) \mid 1 \leq i \leq m\}$,
  $yd(t) = yd(t_1) \dots yd(t_m)$, and
  $pos(t) = \{ i\alpha \mid 1 \leq i \leq m, \alpha \in pos(t_i) \}$.

For each $t \in T_\Sigma(X)$ and $\alpha \in pos(t)$, we introduce the subterm $t/\alpha \in T_\Sigma(X)$ of t at α and define the label $lab(t, \alpha) \in \Sigma \cup X$ in t at α as follows:

- for $t \in \Sigma_0 \cup X$, $t/\lambda = t$ and $lab(t, \lambda) = t$;
- for $t = f(t_1, \dots, t_m)$ with $m \geq 1$ and $f \in \Sigma_m$, if $\alpha = \lambda$ then $t/\alpha = t$ and $lab(t, \alpha) = f$, otherwise, if $\alpha = i\beta$ with $1 \leq i \leq m$, then $t/\alpha = t_i/\beta$ and $lab(t, \alpha) = lab(t_i, \beta)$.

Let $t \in T_\Sigma(X)$. We call a position $\alpha \in pos(t)$ of t a variable position if $lab(t, \alpha) \in X$. The set of variable positions of t is denoted by $vpos(t)$. That is, $vpos(t) = \{ \alpha \in pos(t) \mid lab(t, \alpha) \in X \}$. Furthermore, $root(t) = lab(t, \lambda)$.

For trees $t \in T_\Sigma(X_m)$, and $t_1, \dots, t_m \in T_\Sigma(X)$, we denote by $t[t_1, \dots, t_m]$ the tree obtained by substituting t_i for every occurrence of x_i in t, for $1 \leq i \leq m$. A tree language L is a subset of T_Σ.

For any $m \geq 0$, we distinguish a subset $\overline{T}_\Sigma(X_m)$ of $T_\Sigma(X_m)$ as follows: a tree $t \in T_\Sigma(X_m)$ is in $\overline{T}_\Sigma(X_m)$ if and only if $yd(t) = x_1 \dots x_m$.

For each integer $k \geq 0$, we say that a tree $t \in T_\Sigma(X)$ is a k-normal tree over Σ if t satisfies Conditions 1 and 2 [6].

1. $t \in \overline{T}_\Sigma(X_m)$ for some $m \geq 0$.
2. For every $\alpha \in pos(t)$, ($length(\alpha) = k$ and $lab(t, \alpha) \in X_m$) or ($length(\alpha) < k$ and $lab(t, \alpha) \in \Sigma$).

Note that for each k-normal tree t, $height(t) \leq k$ and that the only 0-normal tree is x_1. The set of k-normal trees over Σ is denoted by $NORM_{\Sigma,k}$.

We illustrate our concepts and results via a running example which we present as a sequence of examples throughout Sections 2 and 3. So the ranked alphabet in all examples will be the one introduced below.

Example 1. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{\#\}$, $\Sigma_1 = \{f\}$, and $\Sigma_2 = \{g\}$. The following trees are 3-normal trees over Σ: $\#$, $f(\#)$, $g(\#, g(\#, \#))$, $g(g(\#, g(x_1, x_2)), \#)$, $g(\#, g(\#, f(x_1)))$, $g(f(g(x_1, x_2)), g(f(x_3), \#))$.

By the definition of a k-normal tree we have the following.

Remark 1. For each $k \geq 0$, $\{t \in NORM_{\Sigma,k} \mid height(t) < k\} \subseteq T_\Sigma$.
For all $0 \leq i \leq j \leq k$,
$\{t \in NORM_{\Sigma,k} \mid height(t) \leq i\} \subseteq \{t \in NORM_{\Sigma,k} \mid height(t) \leq j\}$.

Let Σ be a ranked alphabet, $u \in NORM_{\Sigma,k} \cap \overline{T}_\Sigma(X_m)$, $k, m \geq 0$, and $v \in T_\Sigma$. We say that u is a k-normal prefix of v if $v = u[u_1, \ldots, u_m]$ for some $u_1, \ldots, u_m \in T_\Sigma$.

Proposition 1. [6] *For each tree $s \in T_\Sigma$ and $k \geq 0$, s has exactly one k-normal prefix.*

Remark 2. Let $k \geq 1$, $s, t \in T_\Sigma$, and assume that for any $\alpha \in pos(s)$, if $length(\alpha) \leq k - 1$ then $\alpha \in pos(t)$ and $lab(s, \alpha) = lab(t, \alpha)$. Then the k-normal prefix of s is equal to the k-normal prefix of t.

Example 2. Let $s = g(f(\#), f(g(\#, \#)))$. Then
$g(x_1, x_2)$ is the 1-normal prefix of s,
$g(f(x_1), f(x_2))$ is the 2-normal prefix of s,
$g(f(\#), f(g(x_1, x_2))$ is the 3-normal prefix of s, and
s is the k-normal prefix of s for $k \geq 4$.

For $t \in T_\Sigma$, $\alpha \in pos(t)$, and $r \in T_\Sigma$, we define $t[\alpha \leftarrow r] \in T_\Sigma$ as follows.

- If $\alpha = \lambda$, then $t[\alpha \leftarrow r] = r$.
- If $\alpha = i\beta$, for some $i \in N$ and $\beta \in N^*$, then $t = f(t_1, \ldots, t_m)$ with $f \in \Sigma_m$ and $1 \leq i \leq m$. Then $t[\alpha \leftarrow r] = f(t_1, \ldots, t_{i-1}, t_i[\beta \leftarrow r], t_{i+1}, \ldots, t_m)$.

An alphabet Δ is any finite nonempty set, Δ^* stands for the set of words over Δ, and λ denotes the empty word. For an alphabet Δ, we consider the ranked alphabet $\Delta \cup \{\#\}$, where $\# \notin \Delta$. Here each element of Δ is a unary symbol and $\#$ is a nullary symbol. Then we consider a tree in $T_{\Delta \cup \{\#\}}$ as a word over the alphabet $\Delta \cup \#$. For example, let $\Delta = \{a, b\}$. Then the tree $a(b(b(a(\#))))$ is written as the word $abba\#$. Conversely, for each word $w \in \Delta^*$, the word $w\#$ over the alphabet $\Delta \cup \{\#\}$ can be considered as a tree over the ranked alphabet $\Delta \cup \{\#\}$. For example, the word $aab\#$ can be considered as the tree $a(a(b(\#)))$.

2.2 Term Rewrite Systems

Let $\rightarrow \subseteq A \times A$ be a binary relation on a set A. We denote by $\rightarrow^*$ the reflexive, transitive closure of $\rightarrow$.

Let Σ be a ranked alphabet. Then a term rewrite system (TRS) R over Σ is a finite subset of $(T_\Sigma(X) - X) \times T_\Sigma(X)$ such that for each $(l, r) \in R$, each variable of r also occurs in l. Elements (l, r) of R are called rules and are denoted by $l \rightarrow r$. We call l the left-hand side and r the right-hand of the rule $l \rightarrow r$. The set of left-hand sides (resp. right-hand sides) of rules in R is denoted by $lhs(R)$ (resp. $rhs(R)$).

A TRS R is left-linear (resp. right-linear) if each element of $lhs(R)$ (resp. $rhs(R)$) is linear. A left-linear and right-linear TRS R is called linear. A TRS R is ground if each element of $lhs(R) \cup rhs(R)$ is a ground term.

Let R be a TRS over Σ. For any terms $s, t \in T_\Sigma(X)$, position $\alpha \in pos(s)$, and rule $l \rightarrow r$ in R with $l, r \in T_\Sigma(X_m)$, $m \geq 0$, we say that s rewrites to t applying the rule $l \rightarrow r$ at α, and denote this by $s \rightarrow_{\alpha, l \rightarrow r} t$ if there are $s_1, \ldots, s_m \in T_\Sigma(X)$ such that $s/\alpha = l[s_1, \ldots, s_m]$ and $t = s[\alpha \leftarrow r[s_1, \ldots, s_m]]$. Here we also say that s rewrites to t and denote this by $s \rightarrow_R t$.

A sequence

$$s_0 \rightarrow_{\beta_1, l_1 \rightarrow r_1} s_1 \rightarrow_{\beta_2, l_2 \rightarrow r_2} s_2 \rightarrow_{\beta_3, l_3 \rightarrow r_3} \cdots \rightarrow_{\beta_n, l_n \rightarrow r_n} s_n, \; n \geq 0 \qquad (1)$$

is called a reduction sequence with R.

Dauchet and De Comité [4] introduced the inside-out and outside-in one-pass reductions with a TRS R in an intuitive way and illustrated these concepts by examples. Intuitively, for any terms $s, t \in T_\Sigma(X)$, we say that s is rewritten to t in one pass if we rewrite s into t applying some rules such that the left-hand sides do not overlap. Moreover, in case of an OI pass, we rewrite from the outermost of a bracketed expression of a term to the innermost, i.e., in a top-down order, hence the subtrees are rewritten after duplicating subtrees. In case of an IO pass, we rewrite from the innermost of a bracketed expression of a term to the outermost, i.e., in a bottom-up order, hence the subtrees are rewritten before duplicating subtrees.

We now formally define these concepts. An outside-in one-pass (OI) reduction sequence with R is a sequence

$$s_0 \longrightarrow_{\alpha_1, \beta_1, l_1 \rightarrow r_1} s_1 \longrightarrow_{\alpha_2, \beta_2, l_2 \rightarrow r_2} s_2 \longrightarrow_{\alpha_3, \beta_3, l_3 \rightarrow r_3} \cdots \longrightarrow_{\alpha_n, \beta_n, l_n \rightarrow r_n} s_n, \qquad (2)$$

where Conditions 1–4 hold.

1. $n \geq 0$, $s_0, \ldots, s_n \in T_\Sigma(X)$, and $\alpha_i \in pos(s_0)$, $\beta_i \in pos(s_{i-1})$ for $i = 1, \ldots, n$.
2. $s_0 \rightarrow_{\beta_1, l_1 \rightarrow r_1} s_1 \rightarrow_{\beta_2, l_2 \rightarrow r_2} s_2 \rightarrow_{\beta_3, l_3 \rightarrow r_3} \cdots \rightarrow_{\beta_n, l_n \rightarrow r_n} s_n$ is a reduction sequence with R.
3. $\alpha_1 = \beta_1$.
4. For any $1 < j \leq n$, if there is $1 \leq i < j$ such that $\beta_i \preceq \beta_j$, then let k be the largest such i, and then

a) $\alpha_k\gamma\xi = \alpha_j$, for some $\gamma \in vpos(l_k)$ and $\xi \in N^*$,
b) $\beta_k\delta\xi = \beta_j$ for some $\delta \in vpos(r_k)$, and
c) $lab(l_k, \gamma) = lab(r_k, \delta) \in X$.

Informally, along (2), we rewrite in a top-down order, and keep on rewriting the unprocessed subtrees of the initial tree s_0. At the same time we keep track of the positions of the subtrees of the initial tree s_0. For each $i = 1, \ldots, n$, the position α_i points to the subtree s_0/α_i of s_0, and the position β_i points to an occurrence of s_0/α_i in s_{i-1}, to be rewritten in the ith step of (2).

Formally, the terms $s_0, \ldots, s_n$ are called OI sentential forms of s_0. For each term $s \in T_\Sigma(X)$, $SFOI(s)$ is the set of all OI sentential forms of s. That is, $SFOI(s)$ is the set of all terms t such that there is an OI reduction sequence (2) with $s = s_0$ and $t = s_n$. For a tree language $L \subseteq T_\Sigma$, we put

$$SFOI(L) = \bigcup (SFOI(s) \mid s \in L).$$

We usually write (2) in the form

$$s_0 \underset{R}{\rightarrow} s_1 \underset{R}{\rightarrow} \cdots \underset{R}{\rightarrow} s_n, \tag{3}$$

and note that (3) is an OI reduction. The notation

$$s_0 \underset{R,OI}{\Rightarrow} s_n$$

means that there is an OI reduction (3).

Example 3. Consider the left-linear TRS

$$R = \{\, g(x_1, x_2) \rightarrow f(x_2),\ g(x_1, x_2) \rightarrow g(x_1, x_1) \,\}$$

over Σ. Then

$g(g(g(\#,\#),\#),g(\#,\#)) \longrightarrow_{\lambda,\lambda,g(x_1,x_2)\rightarrow g(x_1,x_1)}$
$g(g(g(\#,\#),\#),g(g(\#,\#),\#)) \longrightarrow_{1,1,g(x_1,x_2)\rightarrow f(x_2)}$
$g(f(\#),g(g(\#,\#),\#)) \longrightarrow_{11,21,g(x_1,x_2)\rightarrow f(x_2)} g(f(\#),g(f(\#),\#))$

is an OI reduction sequence with R.

Furthermore,

$SFOI(g(g(\#,\#),\#)) =$
$\{\, g(g(\#,\#),\#), f(\#), g(g(\#,\#),g(\#,\#)),$
$g(f(\#),\#), g(f(\#),g(\#,\#)), g(g(\#,\#),f(\#)), g(f(\#),f(\#)) \,\}$.

An inside-out one-pass (IO) reduction sequence with R is a reduction sequence (1) where Conditions 1 and 2 hold.

1. $s_0, \ldots, s_n \in T_\Sigma(X)$, and $\beta_i \in pos(s_{i-1}) \cap pos(s_0)$ for $i = 1, \ldots, n$,
2. For any $2 \leq j \leq n$, $\{\beta_1, \ldots, \beta_{j-1}\} \cap$
   $(\{\, \gamma \in N^* \mid \gamma \preceq \beta_j \,\} \cup \{\, \beta_j\xi \mid \xi \in (pos(l_j) - vpos(l_j)) \,\}) = \emptyset$.

Informally, Condition 2 ensures that we rewrite in a bottom-up order and that the left-hand sides do not overlap. It says that for each $2 \leq j \leq n$, the positions $\beta_1, \ldots, \beta_{j-1}$ are no prefixes of β_j nor are positions of any nonvariable symbol in the occurrence of the left-hand side l_j of the rule $l_j \to r_j$ when applying it at β_j in the jth step.

The terms $s_0, \ldots, s_n$ are called IO sentential forms of s_0. For each term $s \in T_\Sigma(X)$, $SFIO(s)$ denotes the set of all IO sentential forms of s. For a tree language $L \subseteq T_\Sigma$, let $SFIO(L) = \bigcup(SFIO(s) \mid s \in L)$. We usually write (1) in the form (3) and note that (3) is an IO reduction. The notation $s_0 \Rightarrow_{R,IO} s_n$ means that there is an IO reduction (1).

Example 4. $g(g(g(\#,\#),\#),g(\#,\#)) \to_{11,g(x_1,x_2)\to f(x_2)}$
$g(g(f(\#),\#),g(\#,\#)) \to_{1,g(x_1,x_2)\to f(x_2)} g(f(\#),g(\#,\#))$
is an IO reduction sequence with R. Furthermore,
$g(\#,g(f(\#),\#)) \to_{2,g(x_1,x_2)\to f(x_2)} g(\#,f(\#)) \to_{\lambda,g(x_1,x_2)\to g(x_1,x_1)} g(\#,\#)$
is another IO reduction sequence with R.

Let R be a TRS over Σ, and $s, t \in T_\Sigma(X)$ be arbitrary. We say that s and t are OI joinable for R if $SFOI(s) \cap SFOI(t) \neq \emptyset$. Furthermore, we say that s is an OI ancestor of t with respect to R if $s \Rightarrow_{R,OI} t$. For tree languages L and M over Σ, we say that L and M are OI joinable for R if $SFOI(L) \cap SFOI(M) \neq \emptyset$. For tree languages L and M over Σ, we say that L and M have a common OI ancestor with respect to R if there is a term $t \in T_\Sigma(X)$ such that $SFOI(t) \cap L \neq \emptyset$ and $SFOI(t) \cap M \neq \emptyset$.

We define the IO counterparts of the above definitions replacing OI by IO.

For the definition of the second order OI (resp. IO) inclusion problem, the second order OI (resp. IO) reachability problem, the second order OI (resp. IO) joinability problem, and second order OI (resp. IO) common ancestor problem, see the Introduction.

2.3 Post Correspondence Problem

A Post Correspondence System (PCS for short) over an alphabet Δ is a pair $\langle \mathbf{w}, \mathbf{z} \rangle = \langle (w_1, \ldots, w_n), (z_1, \ldots, z_n) \rangle$, $n \geq 1$, of lists of nonempty words over the alphabet Δ. We say that the index sequence $k_1, \ldots, k_\ell$ with $\ell \geq 1$, $1 \leq k_1, \ldots, k_\ell \leq n$, is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$, if

$$w_{k_1} \ldots w_{k_\ell} = z_{k_1} \ldots z_{k_\ell} .$$

cf. [11]. The Post Correspondence Problem is the question whether or not a given PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ has a solution.

Proposition 2. [11] *The Post Correspondence Problem is unsolvable. That is, there is no algorithm which takes a PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ as input and determines whether or not there is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$.*

2.4 Recognizable Tree Languages

Let Σ be a ranked alphabet, a bottom-up tree automaton (bta) over Σ is a quadruple $\mathcal{A} = (\Sigma, A, R, A_f)$, where A is a finite set of states of rank 0, $\Sigma \cap A = \emptyset$, $A_f (\subseteq A)$ is the set of final states, R is a finite set of rules of the form

$f(a_1, \ldots, a_m) \to a$ with $m \geq 0$, $f \in \Sigma_m$, $a_1, \ldots, a_m, a \in A$.

We call $f(a_1, \ldots, a_m)$ the left-hand side of the rule $f(a_1, \ldots, a_m) \to a$. We consider R as a ground TRS over $\Sigma \cup A$. The tree language recognized by $\mathcal{A}$ is $L(\mathcal{A}) = \{ t \in T_\Sigma \mid (\exists a \in A_f)\ t \to_R^* a \}$. We say that a tree language L is recognizable if there exists a bta $\mathcal{A}$ such that $L(\mathcal{A}) = L$ [7]. The bta $\mathcal{A} = (\Sigma, A, R, A_f)$ is total if for all $f \in \Sigma_m$, $m \geq 0$, and $a_1, \ldots, a_m$, R has a rule with the left-hand side $f(a_1, \ldots, a_m)$. The bta $\mathcal{A} = (\Sigma, A, R, A_f)$ is deterministic (dbta) if R has no two rules with the same left-hand side. We give a recognizable tree language L via a total dbta $\mathcal{A}$ recognizing L. Let $\mathcal{A}$ be a total dbta. Then for each tree $t \in T_\Sigma$, there is exactly one state $a \in A$ such that $t \to_{\mathcal{A}}^* a$. We denote this a by $t^{\mathcal{A}}$.

Proposition 3. *Let $\mathcal{A} = (\Sigma, A, R_{\mathcal{A}}, A_f)$ be an arbitrary total dbta. Let $s, u \in T_\Sigma$ and $\alpha \in pos(s)$. If $(s/\alpha)^{\mathcal{A}} = u^{\mathcal{A}}$, then $s^{\mathcal{A}} = (s[\alpha \leftarrow u])^{\mathcal{A}}$.*

In the rest of the paper we write $s[\alpha \leftarrow u]^{\mathcal{A}}$ for $(s[\alpha \leftarrow u])^{\mathcal{A}}$.

3 OI One Pass Reductions

We show that for left-linear TRSs, the second order OI inclusion problem and the second order OI reachability problem are decidable.

Theorem 1. *For any left-linear TRS R and recognizable tree languages L and M over Σ, it is decidable whether $SFOI(L) \subseteq M$ and whether $SFOI(L) \cap M \neq \emptyset$.*

The proof needs a long preparation, we now start the process of getting ready for it. Let the total dbtas $\mathcal{A} = (\Sigma, A, R_{\mathcal{A}}, A_f)$ and $\mathcal{B} = (\Sigma, B, R_{\mathcal{B}}, B_f)$ be such that $L(\mathcal{A}) = L$ and $L(\mathcal{B}) = M$. We introduce the ranked alphabet $\Delta = \Sigma \times A \times \mathcal{P}(B)$. We write the elements of Δ in the form $\langle f, a, C \rangle$ and the rank of a symbol $\langle f, a, C \rangle$ in Δ is the rank of f in Σ. To every $s \in T_\Sigma$, we associate an element of Δ, denoted by $val(s)$ as follows: let $val(s) = \langle f, a, C \rangle$, where $root(s) = f$, $s^{\mathcal{A}} = a$, and

$$C = \{ b \mid b = t^{\mathcal{B}} \text{ for some } t \in SFOI(s) \}.$$

Let $s \in T_\Sigma$ be arbitrary. The evaluated copy of s, denoted by $ec(s)$, is a term in T_Δ defined in the following way.

- $pos(ec(s)) = pos(s)$,
- for each $\alpha \in pos(s)$, $lab(ec(s), \alpha) = val(s/\alpha)$.

For each $k \geq 0$, the evaluated k-prefix of s is the k-normal prefix of $ec(s)$, and is denoted by $ep_k(s)$.

Remark 3. For any $s \in T_\Sigma$ and $k \geq 0$, $ep_k(s) \in NORM_{\Delta,k}$.

Example 5. Let $L(\mathcal{A}) = L$, where $\mathcal{A} = (\Sigma, A, R_\mathcal{A}, A_f)$, $A = \{0,1\}$, $A_f = \{0\}$, and $R_\mathcal{A}$ consists of the rules

$\# \to 1$,
$f(0) \to 0$, $f(1) \to 1$,
$g(0,0) \to 0$, $g(0,1) \to 1$, $g(1,0) \to 1$, $g(1,1) \to 0$.

Let $L(\mathcal{B}) = M$, where $\mathcal{B} = (\Sigma, B, R_\mathcal{B}, B_f)$, $B = \{0,1,2\}$, $B_f = \{0\}$, and $R_\mathcal{B}$ consists of the rules

$\# \to 1$,
$f(0) \to 0$, $f(1) \to 1$, $f(2) \to 2$,
$g(0,0) \to 0$, $g(0,1) \to 1$, $g(0,2) \to 2$,
$g(1,0) \to 1$, $g(1,1) \to 2$, $g(1,2) \to 0$,
$g(2,0) \to 2$, $g(2,1) \to 0$, $g(2,2) \to 1$.

Then

$val(g(\#,\#)) = \langle g, 0, \{1,2\}\rangle$,
$val(g(g(\#,\#),\#)) = \langle g, 1, \{0,1,2\}\rangle$,
$ep_1(g(g(\#,\#),\#)) = \langle g, 1, \{0,1,2\}\rangle(x_1, x_2)$,
$ep_2(g(g(\#,\#),\#)) =$
$\langle g, 1, \{0,1,2\}\rangle(\langle g, 0, \{1,2\}\rangle(x_1, x_2), \langle \#, 1, \{1\}\rangle)$,
$ep_2(g(g(g(\#,\#),\#),\#)) =$
$\langle g, 0, \{0,1,2\}\rangle(\langle g, 1, \{0,1,2\}\rangle(x_1, x_2), \langle \#, 1, \{1\}\rangle)$,
$ep_3(g(g(g(\#,\#),\#),\#)) =$
$\langle g, 0, \{0,1,2\}\rangle(\langle g, 1, \{0,1,2\}\rangle(\langle g, 0, \{1,2\}\rangle(x_1, x_2), \langle \#, 1, \{1\}\rangle), \langle \#, 1, \{1\}\rangle)$,
$ep_4(g(g(g(\#,\#),\#),\#)) =$
$\langle g, 0, \{0,1,2\}\rangle(\langle g, 1, \{0,1,2\}\rangle(\langle g, 0, \{1,2\}\rangle(\langle \#, 1, \{1\}\rangle, \langle \#, 1, \{1\}\rangle)$,
$\langle \#, 1, \{1\}\rangle), \langle \#, 1, \{1\}\rangle)$.

Let

$$\tau = max\{\, height(l) \mid l \in lhs(R)\,\} + 1. \tag{4}$$

Lemma 1. *For any $s, t \in T_\Sigma$ and $\mu \in pos(s)$, if $ep_\tau(s/\mu) = ep_\tau(t)$, then $root(s) = root(s[\mu \leftarrow t])$, $s^\mathcal{A} = s[\mu \leftarrow t]^\mathcal{A}$, and*

$$\{\, u^\mathcal{B} \mid u \in SFOI(s)\,\} \subseteq \{\, u^\mathcal{B} \mid u \in SFOI(s[\mu \leftarrow t])\,\}. \tag{5}$$

Proof. Let $s, t \in T_\Sigma$, and $\mu \in pos(s)$ such that

$$ep_\tau(s/\mu) = ep_\tau(t).$$

If $\mu = \lambda$, then $s/\mu = s$ and $t = s[\mu \leftarrow t]$. By our assumption, $ep_\tau(s) = ep_\tau(s[\mu \leftarrow t])$. Consequently, the statement of the lemma holds.

From now on, we assume that $\mu \neq \lambda$. Then $root(s) = root(s[\mu \leftarrow t])$ and $s^\mathcal{A} = s[\mu \leftarrow t]^\mathcal{A}$ by Proposition 3. It is left to show (5). To this end, let $b \in \{\, u^\mathcal{B} \mid u \in SFOI(s)\,\}$. Then there is an OI reduction sequence

(a) $s \longrightarrow_{\bar{\alpha}_1, \bar{\beta}_1, \bar{l}_1 \to \bar{r}_1} \bar{s}_1 \longrightarrow_{\bar{\alpha}_2, \bar{\beta}_2, \bar{l}_2 \to \bar{r}_2} \cdots \longrightarrow_{\bar{\alpha}_n, \bar{\beta}_n, \bar{l}_n \to \bar{r}_n} \bar{s}_n$

such that

$$b = \bar{s}_n^{\mathcal{B}}.$$

Along (a), R may have produced more than one copies of the subtree s/μ of s. We now consider the rewriting of the ℓth copy of s/μ. R may rewrite at some $\alpha \in pos(s)$ such that

$$\alpha \prec \mu$$

and the left-hand side of the applied rule $l \to r$ overlaps with the ℓth copy of s/μ. Then R rewrites the subtrees $s/\chi_1, \ldots, s/\chi_k$, where $\chi_1, \ldots, \chi_k \in pos(s)$, of the ℓth copy of s/μ. There are

$$\nu_{\ell 1}, \ldots, \nu_{\ell n_\ell} \in \{\, \alpha\xi \mid \xi \in vpos(l) \,\} \tag{6}$$

such that for each χ_j, $j = 1, \ldots, k$, there is a $\nu_{\ell i}$, $i \in \{1, \ldots, n_\ell\}$, such that

$$\mu \preceq \nu_{\ell i} \preceq \chi_j. \tag{7}$$

Thus $\nu_{\ell i}$ is an extension of μ and a prefix of χ_j. Then by $\alpha \prec \mu$, (4), (6), and (7),

$$\nu_{\ell i} = \mu\eta_{\ell i} \text{ for some } \eta_{\ell i} \in N^* \text{ with } length(\eta_{\ell i}) \leq \tau - 2 \text{ for each } i = 1, \ldots, n_\ell. \tag{8}$$

Note that we do not necessarily rewrite at the positions $\nu_{\ell 1}, \ldots, \nu_{\ell n_\ell}$, rather we rewrite at the extensions $\chi_1, \ldots, \chi_k$ of the positions $\nu_{\ell 1}, \ldots, \nu_{\ell n_\ell}$. The positions $\nu_{11}, \ldots, \nu_{1n_1}, \nu_{21}, \ldots, \nu_{2n_2}, \ldots$ are simply denoted by $\nu_1, \ldots, \nu_m$. We rearrange the rewrite steps of the OI reduction sequence (a) into the OI reduction sequence (b) below such that the following conditions hold.

- Beginning with step $d_0 + 1$, we carry out the reduction steps that take place at the extensions of μ. Intuitively, the suffix of (b), starting at step $d_0 + 1$, consists of the reduction of the copies of s/μ.
- When rewriting the ℓth copy of s/μ, first we carry out all reduction steps at the extensions of $\nu_{\ell 1}$, second we carry out all reduction steps at the extensions of $\nu_{\ell 2}$, and so on. Thus we have

(b) $s \longrightarrow_{\alpha_1,\beta_1,l_1\to r_1} s_1 \longrightarrow_{\alpha_2,\beta_2,l_2\to r_2} s_2 \longrightarrow_{\alpha_3,\beta_3,l_3\to r_3} \cdots \longrightarrow_{\alpha_{d_0},\beta_{d_0},l_{d_0}\to r_{d_0}}$
$s_{d_0} = v[s/\nu_1, \ldots, s/\nu_m] \longrightarrow_{\alpha_{d_0+1},\beta_{d_0+1},l_{d_0+1}\to r_{d_0+1}} \cdots \longrightarrow_{\alpha_{d_1},\beta_{d_1},l_{d_1}\to r_{d_1}}$
$v[v_1, s/\nu_2, \ldots, s/\nu_m] \longrightarrow_{\alpha_{d_1+1},\beta_{d_1+1},l_{d_1+1}\to r_{d_1+1}} \cdots \longrightarrow_{\alpha_{d_m},\beta_{d_m},l_{d_m}\to r_{d_m}}$
$v[v_1, v_2, \ldots, v_m] = s_n,$
where

- $v \in \overline{T}(X_m)$, $m \geq 0$,
- $\vartheta_i \in vpos(v)$ and $lab(v, \vartheta_i) = x_i$ for $i = 1, \ldots, m$,
- $0 \leq d_0 < d_1 < \cdots < d_m = n$,
- $\bar{s}_n = s_n$.

Moreover, the following four conditions hold.

1. For each $1 \leq i \leq d_0$, μ is not a prefix of α_i, and for each $d_0 + 1 \leq i \leq d_m$, $\mu \preceq \alpha_i$.
2. For each $1 \leq i \leq m$, we have $\nu_i = \mu\zeta_i$ for some $\zeta_i \in N^*$ with $length(\zeta_i) \leq \tau - 2$.
3. For each $1 \leq i \leq m$ and $d_{i-1} + 1 \leq j \leq d_i$, we have $\nu_i \preceq \alpha_j$ and $\vartheta_i \preceq \beta_j$.
4. For each $1 \leq i \leq m$, there is an OI reduction sequence $s/\nu_i \Rightarrow_{R,OI} v_i$ for some subtree v_i of s_n.

We obtained Condition 2 by (8). Since $b = \bar{s}_n^{\mathcal{B}}$ and $\bar{s}_n = s_n$, we also have

$$b = s_n^{\mathcal{B}}.$$

Note that along the reduction in Condition 4 we do not necessarily rewrite s/ν_i at the position λ. By Condition 2 and the assumption $ep_\tau(s/\mu) = ep_\tau(t)$ of the lemma, $val(s/\nu_i) = val(s[\mu \leftarrow t]/\nu_i)$. Hence by Condition 4, for each $1 \leq i \leq m$, there is a tree $z_i \in T_\Sigma$ such that

$$s[\mu \leftarrow t]/\nu_i \underset{R,OI}{\Rightarrow} z_i \text{ and } z_i^{\mathcal{B}} = v_i^{\mathcal{B}}.$$

Hence, as R is left-linear, we have the OI reduction sequence
$s[\mu \leftarrow t] \longrightarrow_{\alpha_1,\beta_1,l_1\to r_1} s_1' \longrightarrow_{\alpha_2,\beta_2,l_2\to r_2} s_2' \longrightarrow_{\alpha_3,\beta_3,l_3\to r_3} \cdots \longrightarrow_{\alpha_{d_0},\beta_{d_0},l_{d_0}\to r_{d_0}}$
$s_{d_0}' = v[s[\mu \leftarrow t]/\nu_1, \ldots, s[\mu \leftarrow t]/\nu_m] \longrightarrow_{\gamma_{d_0+1},\delta_{d_0+1},l_{d_0+1}\to r_{d_0+1}} \cdots$
$\longrightarrow_{\gamma_{e_1},\delta_{e_1},l_{e_1}\to r_{e_1}} v[z_1, s[\mu \leftarrow t]/\nu_2, \ldots, s[\mu \leftarrow t]/\nu_m] \longrightarrow_{\gamma_{e_1+1},\delta_{e_1+1},l_{e_1+1}\to r_{e_1+1}} \cdots$
$\longrightarrow_{\gamma_{e_m},\delta_{e_m},l_{e_m}\to r_{e_m}} v[z_1, z_2, \ldots, z_m]$.
Here

$$d_0 < e_1 < \cdots < e_m,$$

and the following conditions hold.

- For each $1 \leq i \leq d_0$, $s_i' \in T_\Sigma$.
- For each $1 \leq i \leq d_0$, μ is not a prefix of α_i, and for each $d_0 + 1 \leq i \leq e_m$, $\mu \preceq \gamma_i$.
- For each $d_0 + 1 \leq j \leq e_1$, we have $\nu_1 \preceq \gamma_j$ and $\vartheta_1 \preceq \delta_j$. Furthermore, for each $2 \leq i \leq m$ and $e_{i-1} + 1 \leq j \leq e_i$, we have $\nu_i \preceq \gamma_j$ and $\vartheta_i \preceq \delta_j$.
- For each $1 \leq i \leq m$, there is an OI reduction sequence $s[\mu \leftarrow t]/\nu_i \Rightarrow_{R,OI} z_i$.

By the definition of z_i, $i = 1, \ldots, m$, and Proposition 3,

$$s_n^{\mathcal{B}} = v[v_1, \ldots, v_m]^{\mathcal{B}} = v[z_1, \ldots, z_m]^{\mathcal{B}}.$$

Thus, by $b = s_n^{\mathcal{B}}$, we have $b \in \{ u^{\mathcal{B}} \mid u \in SFOI(s[\mu \leftarrow t]) \}$. This proves (5), and with it also finishes the proof of the lemma. □

Lemma 2. *For any* $s, t \in T_\Sigma$ *and* $\mu \in pos(s)$, *if* $ep_\tau(s/\mu) = ep_\tau(t)$, *then* $\{ u^{\mathcal{B}} \mid u \in SFOI(s) \} = \{ u^{\mathcal{B}} \mid u \in SFOI(s[\mu \leftarrow t]) \}$.

Proof. The inclusion from left to right is proven in Lemma 1. The other inclusion can be verified by applying Lemma 1 on $s' = s[\mu \leftarrow t]$ and $t' = s/\mu$. □

By Lemma 1 and Lemma 2 we have the following result.

Lemma 3. *For any* $s, t \in T_\Sigma$ *and* $\mu \in pos(s)$, *if* $ep_\tau(s/\mu) = ep_\tau(t)$, *then* $val(s) = val(s[\mu \leftarrow t])$.

Lemma 4. *For any* $s, t \in T_\Sigma$ *and* $\mu \in pos(s)$, *if* $ep_\tau(s/\mu) = ep_\tau(t)$, *then* $ep_\tau(s) = ep_\tau(s[\mu \leftarrow t])$.

Proof. Let $s, t \in T_\Sigma$ and $\mu \in pos(s)$ such that

$$ep_\tau(s/\mu) = ep_\tau(t). \tag{9}$$

Let $\alpha \in pos(s)$ such that $length(\alpha) \leq \tau - 1$. We now show that

$$lab(ec(s), \alpha) = lab(ec(s[\mu \leftarrow t]), \alpha). \tag{10}$$

For this we distinguish the following three cases.

Case 1: α is not a prefix of μ and μ is not a prefix of α. In this case, $s/\alpha = s[\mu \leftarrow t]/\alpha$, and hence we have (10).

Case 2: $\alpha \preceq \mu$. Then by (9) and Lemma 3, (10) holds.

Case 3: $\mu \prec \alpha$. Then $\alpha = \mu\beta$ for some $\beta \in pos(s/\mu)$, and

$lab(ec(s), \alpha) = lab(ec(s/\mu), \beta) =$

$lab(ec(t), \beta) = lab(ec(s[\mu \leftarrow t]), \alpha)$ (by (9)).

Hence (10) holds.

In the aggregate, we conclude that for each $\alpha \in pos(s)$ with $length(\alpha) \leq \tau - 1$, (10) holds. Consequently, by Remark 2, the τ-normal prefix of $ec(s)$ is equal to the τ-normal prefix of $ec(s[\mu \leftarrow t])$, that is, $ep_\tau(s) = ep_\tau(s[\mu \leftarrow t])$. □

For each $i \geq 0$, let $P_i = \{ ep_\tau(s) \mid s \in T_\Sigma$ and $height(s) \leq i \}$. It should be clear that P_i can be computed effectively for every $i \geq 0$.

Example 6. Observe that $\tau = 2$. The trees in T_Σ with height at most two are the following:

$\#$, $f(\#)$, $g(\#, \#)$,

$f(f(\#))$, $f(g(\#, \#))$, $g(\#, f(\#))$, $g(f(\#), \#)$, $g(f(\#), f(\#))$, $g(\#, g(\#, \#))$,

$g(f(\#), g(\#, \#))$, $g(g(\#, \#), \#)$, $g(g(\#, \#), f(\#))$, $g(g(\#, \#), g(\#, \#))$.

Moreover, $P_0 = \{ \langle \#, 1, \{1\} \rangle \}$,

P_1 consists of the trees:

$\langle \#, 1, \{1\} \rangle$, $\langle f, 1, \{1\} \rangle(\langle \#, 1, \{1\} \rangle)$, $\langle g, 0, \{1, 2\} \rangle(\langle \#, 1, \{1\} \rangle, \langle \#, 1, \{1\} \rangle)$,

and P_2 consists of the trees:

$\langle \#, 1, \{1\} \rangle$, $\langle f, 1, \{1\} \rangle(\langle \#, 1, \{1\} \rangle)$, $\langle g, 0, \{1, 2\} \rangle(\langle \#, 1, \{1\} \rangle, \langle \#, 1, \{1\} \rangle)$,

$\langle f, 1, \{1\} \rangle(\langle f, 1, \{1\} \rangle(x_1))$, $\langle f, 0, \{1, 2\} \rangle(\langle g, 0, \{1, 2\} \rangle(x_1, x_2))$,

$\langle g,0,\{1,2\}\rangle(\langle \#,1,\{1\}\rangle,\langle f,1,\{1\}\rangle(x_1)),$
$\langle g,0,\{1,2\}\rangle(\langle f,1,\{1\}\rangle(x_1),\langle \#,1,\{1\}\rangle),$
$\langle g,0,\{1,2\}\rangle(\langle f,1,\{1\}\rangle(x_1),\langle f,1,\{1\}\rangle(x_2)),$
$\langle g,1,\{0,1,2\}\rangle(\langle \#,1,\{1\}\rangle,\langle g,0,\{1,2\}\rangle(x_1,x_2)),$
$\langle g,1,\{0,1,2\}\rangle(\langle f,1,\{1\}\rangle(x_1),\langle g,0,\{1,2\}\rangle(x_2,x_3)),$
$\langle g,1,\{0,1,2\}\rangle(\langle g,0,\{1,2\}\rangle(x_1,x_2),\langle \#,1,\{1\}\rangle),$
$\langle g,1,\{0,1,2\}\rangle(\langle g,0,\{1,2\}\rangle(x_1,x_2),\langle f,1,\{1\}\rangle(x_3)),$
$\langle g,0,\{0,1,2\}\rangle(\langle g,0,\{1,2\}\rangle(x_1,x_2),\langle g,0,\{1,2\}\rangle(x_3,x_4)).$

By Remark 3 and the definition of P_i, $i \geq 0$,

$$P_0 \subseteq P_1 \subseteq \cdots \subseteq NORM_{\Delta,\tau}. \tag{11}$$

Since $NORM_{\Delta,\tau}$ is a finite set, there is a smallest index n such that $P_n = P_{n+1}$. We call n the evaluation index. From now on, throughout this section, n stands for the evaluation index.

Lemma 5. *$P_n = P_\ell$ for $\ell > n$.*

Proof. We proceed by induction on ℓ.

Base Step: $\ell = n+1$. Then $P_n = P_\ell$ by the definition of the evaluation index n.

Induction step: Let $\ell \geq n+2$, and assume that $P_n = P_j$ for all j such that $n \leq j \leq \ell - 1$.

Let $s \in T_\Sigma$ with $height(s) = \ell$. Then $s = f(s_1, \ldots s_m)$ for some $f \in \Sigma_m$, $m \geq 1$, and $s_1, \ldots s_m \in T_\Sigma$. For each $i = 1, \ldots, m$, we define the tree $v_i \in T_\Sigma$ as follows. If $height(s_i) \leq \ell - 2$, then let $v_i = s_i$. Assume that $height(s_i) = \ell - 1$. By the induction hypothesis, there is a tree $v_i \in T_\Sigma$ such that $height(v_i) \leq \ell - 2$ and $ep_\tau(s_i) = ep_\tau(v_i)$. Let

$$v = f(v_1, \ldots, v_m).$$

Then $height(v) \leq \ell - 1$. Hence, by (11) $ep_\tau(v) \in P_{\ell-1}$. By the induction hypothesis,

$$ep_\tau(v) \in P_n. \tag{12}$$

By Lemma 4,

$ep_\tau(f(s_1, s_2, \ldots, s_m)) = ep_\tau(f(v_1, s_2 \ldots s_m)) =$
$ep_\tau(f(v_1, v_2, s_3, \ldots, s_m)) = \cdots = ep_\tau(f(v_1, v_2, \ldots, v_m)).$

Thus $ep_\tau(s) = ep_\tau(v)$. By (12), $ep_\tau(s) \in P_n$. Since $s \in T_\Sigma$ with $height(s) = \ell$ is arbitrary, we have $P_\ell \subseteq P_n$. By (11) $P_\ell = P_n$. □

Lemma 6. *The evaluation index n can be computed effectively.*

Proof. Let us compute P_i for $i = 0, 1, \ldots$ until we find that $P_i = P_{i+1}$. As mentioned above, the procedure stops at some i. By definition, n equals to this i. □

By the definition of $val(p)$ for $p \in T_\Sigma$, and the definition of P_i, $i \geq 0$, we have the following.

Lemma 7. *$SFOI(L) \subseteq M$ if and only if for each $p \in P_n$ with $p = \langle f, a, C\rangle(p_1, \ldots, p_m)$, $m \geq 0$, if $a \in A_f$ then $C \subseteq B_f$.*
$SFOI(L) \cap M \neq \emptyset$ if and only if there is $p \in P_n$ such that $p = \langle f, a, C\rangle(p_1, \ldots, p_m)$, $m \geq 0$, $a \in A_f$, and $C \cap B_f \neq \emptyset$.

Example 7. We now show that $SFOI(L) \not\subseteq M$ and $SFOI(L) \cap M \neq \emptyset$. To this end we need not compute n and P_n. Observe that

$\langle g, 0, \{1,2\}\rangle(\langle \#, 1, \{1\}\rangle, \langle \#, 1, \{1\}\rangle) \in P_2 \subseteq P_n$, $0 \in A_f$, and $\{1,2\} \not\subseteq B_f$.
Hence by Lemma 7, $SFOI(L) \not\subseteq M$. Furthermore,

$\langle g, 0, \{0,1,2\}\rangle(\langle g, 0, \{1,2\}\rangle(x_1, x_2), \langle g, 0, \{1,2\}\rangle(x_3, x_4)) \in P_2 \subseteq P_n$, $0 \in A_f$, and $\{0,1,2\} \cap B_f \neq \emptyset$.
Hence by Lemma 7, $SFOI(L) \cap M \neq \emptyset$.

Proof. (of Theorem 1) Recall that $\mathcal{A} = (\Sigma, A, R_{\mathcal{A}}, A_f)$ and $\mathcal{B} = (\Sigma, B, R_{\mathcal{B}}, B_f)$ are total dbtas such that $L(\mathcal{A}) = L$ and $L(\mathcal{B}) = M$. Let us compute the evaluation index n and the set P_n (cf. Lemma 6). For each $p \in P_n$ with $p = \langle f, a, C\rangle(p_1, \ldots, p_m)$, $m \geq 0$, by direct inspection we decide whether $a \in A_f$ implies $C \subseteq B_f$. If for each $p \in P_n$, the answer is yes, then $SFOI(L) \subseteq M$. Otherwise, $SFOI(L) \not\subseteq M$, see Lemma 7.

For each $p \in P_n$ with $p = \langle f, a, C\rangle(p_1, \ldots, p_m)$, $m \geq 0$, by direct inspection we decide whether $a \in A_f$ implies $C \cap B_f \neq \emptyset$. If the answer is yes for some $p \in P_n$, then $SFOI(L) \cap M \neq \emptyset$. Otherwise, $SFOI(L) \cap M = \emptyset$, see Lemma 7. □

4 Second Order OI Joinability Problem

We show that the second order OI joinability problem is undecidable for left-linear TRSs.

For an alphabet Δ, we also consider the alphabets $\overline{\Delta} = \{\overline{a} \mid a \in \Delta\}$ and $\widehat{\Delta} = \{\widehat{a} \mid a \in \Delta\}$. The alphabets Δ, $\overline{\Delta}$, and $\widehat{\Delta}$ are pairwise disjoint. For each word $w \in \Delta^*$, the word $\overline{w} \in \overline{\Delta}^*$ is defined as follows.

- If $w = \lambda$, then $\overline{w} = \lambda$.
- If $w = az$ for some $a \in \Sigma$ and $z \in \Delta^*$, then $\overline{w} = \overline{a}\,\overline{z}$.

For each word $w \in \Delta^*$, we define the word $\widehat{w} \in \widehat{\Delta}^*$ in a similar way to $\overline{w}$.

Let $\langle \mathbf{w}, \mathbf{z}\rangle = \langle (w_1, \ldots, w_n), (z_1, \ldots, z_n)\rangle$ be a PCS over the alphabet Δ. We associate the ranked alphabet Σ, the recognizable tree languages L and M over Σ, and the TRS R over Σ with the PCS $\langle \mathbf{w}, \mathbf{z}\rangle$. To this end, we consider the sets Δ, $\overline{\Delta}$, $\widehat{\Delta}$, and $\Gamma = \{1, \ldots, n\}$ of unary symbols. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma_4$, $\Sigma_0 = \{\#\}$, $\Sigma_1 = \Delta \cup \overline{\Delta} \cup \widehat{\Delta} \cup \Gamma$, $\Sigma_2 = \{f, g\}$, $\Sigma_4 = \{h\}$. Let

$$L = \{ f(s,t) \mid s \in (T_{\Gamma \cup \{\#\}} - \{\#\}), t \in T_{\widehat{\Delta} \cup \{\#\}} \}$$

and

$$M = \{ g(s,t) \mid s \in T_{\Delta \cup \{\#\}}, t \in T_{\overline{\Delta} \cup \{\#\}} \}.$$

The TRS R consists of the following rules:

- $f(x_1, x_2) \to h(x_1, x_1, x_2, x_2)$, $g(x_1, x_2) \to h(x_1, x_2, x_1, x_2)$,
- $k(x_1) \to w_k(x_1)$, $k(x_1) \to \overline{z_k}(x_1)$ for $k = 1, \ldots, n$,
- $\widehat{a}(x_1) \to a(x_1)$, $\widehat{a}(x_1) \to \overline{a}(x_1)$ for $a \in \Delta$.

Example 8. Let $\Delta = \{a, b\}$. Let PCS $\langle \mathbf{w}, \mathbf{z} \rangle = \langle (a, ab), (aa, b) \rangle$. Note that 12 is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. The ranked alphabet Σ consists of

- the nullary symbol $\#$,
- the unary symbols $1, 2, a, b, \overline{a}, \overline{b}, \widehat{a}, \widehat{b}$,
- the binary symbols f, g, and
- the symbol h of rank 4.

The TRS R consists of the following rules:

- $f(x_1, x_2) \to h(x_1, x_1, x_2, x_2)$, $g(x_1, x_2) \to h(x_1, x_2, x_1, x_2)$,
- $1x_1 \to ax_1$, $2x_1 \to abx_1$, $1x_1 \to \overline{a}\overline{a}x_1$, $2x_1 \to \overline{b}x_1$,
- $\widehat{a}x_1 \to ax_1$, $\widehat{a}x_1 \to \overline{a}x_1$, $\widehat{b}x_1 \to bx_1$, $\widehat{b}x_1 \to \overline{b}x_1$.

Let

$$L = \{ f(s,t) \mid s \in (T_{\{1,2,\#\}} - \{\#\}), t \in T_{\{\widehat{a},\widehat{b},\#\}} \}$$

and

$$M = \{ g(s,t) \mid s \in T_{\{a,b,\#\}}, t \in T_{\{\overline{a},\overline{b},\#\}} \}.$$

Since 12 is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$, we have the following OI reduction sequence.

$f(12\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(12\#, 12\#, \widehat{a}\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(a2\#, 12\#, \widehat{a}\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R$
$h(aab\#, 12\#, \widehat{a}\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(aab\#, \overline{a}\overline{a}2\#, \widehat{a}\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R$
$h(aab\#, \overline{a}\overline{a}\overline{b}\#, \widehat{a}\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(aab\#, \overline{a}\overline{a}\overline{b}\#, a\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R$
$h(aab\#, \overline{a}\overline{a}\overline{b}\#, a\widehat{a}\widehat{b}\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(aab\#, \overline{a}\overline{a}\overline{b}\#, aab\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R$
$h(aab\#, \overline{a}\overline{a}\overline{b}\#, aab\#, \widehat{a}\widehat{a}\widehat{b}\#) \to_R h(aab\#, \overline{a}\overline{a}\overline{b}\#, aab\#, \overline{a}\overline{a}\widehat{b}\#) \to_R$
$h(aab\#, \overline{a}\overline{a}\overline{b}\#, aab\#, \overline{a}\overline{a}\overline{b}\#)$.

Consider the OI-reduction sequence

$g(aab\#, \overline{a}\overline{a}\overline{b}\#) \to_R h(aab\#, \overline{a}\overline{a}\overline{b}\#, aab\#, \overline{a}\overline{a}\overline{b}\#)$.

Thus, the terms $f(12\#, \widehat{a}\widehat{a}\widehat{b}\#)$ and $g(aab\#, \overline{a}\overline{a}\overline{b}\#)$ are OI joinable. Consequently L and M are OI joinable.

Lemma 8. *The PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ has a solution if and only if L and M are OI joinable.*

Proof. Assume that the index sequence $k_1, \ldots, k_\ell$ is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. Then

$$w_{k_1} \ldots w_{k_\ell} = z_{k_1} \ldots z_{k_\ell}. \tag{13}$$

We have the OI-reduction sequence

(a) $f(k_1 \ldots k_\ell\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R$
$h(k_1 \ldots k_\ell\#, k_1 \ldots k_\ell\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R$
$h(w_{k_1}k_2 \ldots k_\ell\#, k_1 \ldots k_\ell\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, k_1 \ldots k_\ell\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}}k_2 \ldots k_\ell\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, w_{k_1}\widehat{w_{k_2}} \ldots \widehat{w_{k_\ell}}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \widehat{w_{k_1}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \widehat{w_{k_1}}\widehat{w_{k_2}} \ldots \widehat{w_{k_\ell}}\#) \to_R^*$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#).$

We also have the OI-reduction sequence

(b) $g(w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#) \to_R$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#).$

By (13),

$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#) =$
$h(w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#, w_{k_1} \ldots w_{k_\ell}\#, \overline{w_{k_1}} \ldots \overline{w_{k_\ell}}\#).$

Hence $SFOI(L) \cap SFOI(M) \neq \emptyset$. That is, L and M are OI joinable.

Conversely, assume that L and M are OI joinable, i.e., $SFOI(L) \cap SFOI(M) \neq \emptyset$. Consequently, there are $u \in SFOI(L) \cap SFOI(M)$, $1 \leq k_1, \ldots, k_\ell \leq n$, $\ell \geq 1$,

$$s \in T_{\widehat{\Delta} \cup \{\#\}}, t_1 \in \Delta^*, \text{ and } t_2 \in \overline{\Delta}^* \tag{14}$$

such that

$$f(k_1 \ldots k_\ell\#, s) \underset{R,OI}{\Rightarrow} u \tag{15}$$

and

$$g(t_1\#, t_2\#) \underset{R,OI}{\Rightarrow} u. \tag{16}$$

We write (15) in the form

(c) $f(k_1 \ldots k_\ell\#, s) \to_{R,OI} h(k_1 \ldots k_\ell\#, k_1 \ldots k_\ell\#, s, s) \Rightarrow_{R,OI} u = h(u_1\#, u_2\#, u_3\#, u_4\#)$ for some $u_1, u_2, u_3, u_4 \in \Sigma^*$. Here

(d) $k_1 \ldots k_\ell\# \Rightarrow_{R,OI} u_1\#$ and $k_1 \ldots k_\ell\# \Rightarrow_{R,OI} u_2\#$, and

(e) $s \Rightarrow_{R,OI} u_3\#$ and $s \Rightarrow_{R,OI} u_4\#$.

We write (16) in the form

(f) $g(t_1\#, t_2\#) \to_{R,OI} h(t_1\#, t_2\#, t_1\#, t_2\#) = u$. Hence

$$t_1\# = u_1\# = u_3\# \text{ and } t_2\# = u_2\# = u_4\#. \tag{17}$$

Consequently, by (14) and (e), we get that $s = \widehat{u_3}\#$ and $\overline{u_3}\# = u_4\#$. Hence, by (17), we have

$$\overline{t_1} = t_2. \tag{18}$$

By (d) and (17),

$$k_1 \dots k_\ell \# \underset{R,OI}{\Rightarrow} t_1 \# \text{ and } k_1 \dots k_\ell \# \underset{R,OI}{\Rightarrow} t_2 \#.$$

Since $t_1 \in \Delta^*$ and $t_2 \in \overline{\Delta}^*$,

$$w_{k_1} \dots w_{k_\ell} = t_1 \text{ and } \overline{z}_{k_1} \dots \overline{z_{k_\ell}} = t_2. \tag{19}$$

By (18) and (19), we have

$$\overline{w_{k_1}} \dots \overline{w_{k_\ell}} = \overline{z_{k_1}} \dots \overline{z_{k_\ell}}.$$

Consequently (13) holds. Hence the index sequence $k_1, \dots, k_\ell$ is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. □

Theorem 2. *For left-linear TRSs, the second order OI joinability problem is undecidable.*

Proof. Let $\langle \mathbf{w}, \mathbf{z} \rangle$ be a Post Correspondence System over the alphabet Δ. We associated the ranked alphabet Σ, the recognizable tree languages L and M over Σ and the TRS R over Σ with the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. By Lemma 8, the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ has a solution if and only if L and M are OI joinable. By Proposition 2, there is no algorithm which takes a PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ as input and determines whether or not there is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. □

5 Second Order IO Common Ancestor Problem

We show that the second order common IO ancestor problem is undecidable for right-linear TRSs.

Let $\langle \mathbf{w}, \mathbf{z} \rangle = \langle (w_1, \dots, w_n), (z_1, \dots, z_n) \rangle$ be a Post Correspondence System over the alphabet Δ. We associate the ranked alphabet Σ, the recognizable tree languages L and M over Σ, and the TRS R over Σ with the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. To this end, we consider the sets Δ, $\overline{\Delta} = \{ \overline{d} \mid d \in \Delta \}$, $\Gamma = \{ 1, \dots, n \}$, and $\overline{\Gamma} = \{ \overline{1}, \dots, \overline{n} \}$ of unary symbols. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \Sigma_4$, $\Sigma_0 = \{ \# \}$, $\Sigma_1 = \Delta \cup \overline{\Delta} \cup \Gamma \cup \overline{\Gamma}$, $\Sigma_2 = \{ g, h \}$, $\Sigma_4 = \{ f \}$.
The TRS R consists of the following rules:

- $f(x_1, x_1, x_2, x_2) \to g(x_1, x_2)$, $f(x_1, x_2, x_1, x_2) \to h(x_1, x_2)$,
- $kx_1 \to \overline{k}x_1$, $kx_1 \to w_k x_1$, $\overline{k}x_1 \to \overline{z_k}x_1$ for $k = 1, \dots, n$,
- $dx_1 \to \overline{d}x_1$ for $d \in \Delta$.

Note that R is right-linear.
Let $L = \{ g(s,t) \mid s \in (T_{\overline{\Gamma} \cup \{\#\}} - \{\#\}), t \in T_{\overline{\Delta} \cup \{\#\}} \}$ and $M = \{ h(s,t) \mid s \in T_{\Delta \cup \{\#\}}, t \in T_{\overline{\Delta} \cup \{\#\}} \}$.

Example 9. Let $\Delta = \{a, b\}$. Let PCS $\langle \mathbf{w}, \mathbf{z} \rangle = \langle (a, ab), (aa, b) \rangle$. Note that 12 is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. The ranked alphabet Σ consists of

- the nullary symbol $\#$,
- the unary symbols $1, 2, a, b, \overline{1}, \overline{2}, \overline{a}, \overline{b}$,
- the symbol f of rank 4, and the binary symbols g and h.

The TRS R consists of the following rules:

- $f(x_1, x_1, x_2, x_2) \to g(x_1, x_2)$, $f(x_1, x_2, x_1, x_2) \to h(x_1, x_2)$,
- $1x_1 \to ax_1$, $2x_1 \to abx_1$,
- $\overline{1}x_1 \to \overline{a}\overline{a}x_1$, $\overline{2}x_1 \to \overline{b}x_1$,
- $1x_1 \to \overline{1}x_1$, $2x_1 \to \overline{2}x_1$,
- $ax_1 \to \overline{a}x_1$, $bx_1 \to \overline{b}x_1$.

Furthermore

$$L = \{ g(s,t) \mid s \in (T_{\{\overline{1},\overline{2},\#\}} - \{\#\}), t \in T_{\{\overline{a},\overline{b},\#\}} \}$$

and

$$M = \{ h(s,t) \mid s \in T_{\{a,b,\#\}}, t \in T_{\{\overline{a},\overline{b},\#\}} \}.$$

Since 12 is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$, we have the following two IO reduction sequences.

$f(12\#, \overline{12}\#, aab\#, \overline{aa}\overline{b}\#) \to_R f(12\#, \overline{12}\#, aa\overline{b}\#, \overline{aa}\overline{b}\#) \to_R$
$f(12\#, \overline{12}\#, a\overline{aa}\overline{b}\#, \overline{aa}\overline{b}\#) \to_R f(12\#, \overline{12}\#, \overline{aa}\overline{b}\#, \overline{aa}\overline{b}\#) \to_R$
$f(1\overline{2}\#, \overline{12}\#, \overline{aa}\overline{b}\#, \overline{aa}\overline{b}\#) \to_R f(\overline{12}\#, \overline{12}\#, \overline{aa}\overline{b}\#, \overline{aa}\overline{b}\#) \to_R g(\overline{12}\#, \overline{aa}\overline{b}\#)$
and
$f(12\#, \overline{12}\#, aab\#, \overline{aa}\overline{b}\#) \to_R f(1ab\#, \overline{12}\#, aab\#, \overline{aa}\overline{b}\#) \to_R$
$f(aab\#, \overline{12}\#, aab\#, \overline{aa}\overline{b}\#) \to_R f(aab\#, \overline{1}\overline{b}\#, aab\#, \overline{aa}\overline{b}\#) \to_R$
$f(aab\#, \overline{aa}\overline{b}\#, aab\#, \overline{aa}\overline{b}\#) \to_R h(aab\#, \overline{aa}\overline{b}\#)$.

Thus $f(12\#, \overline{12}\#, aab\#, \overline{aa}\overline{b}\#)$ is a common IO ancestor of the terms $g(\overline{12}\#, \overline{aa}\overline{b}\#)$ and $h(aab\#, \overline{aa}\overline{b}\#)$. Consequently, the tree languages L and M have a common IO one-pass ancestor.

Lemma 9. *The PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ has a solution if and only if L and M have a common IO ancestor.*

Proof. Assume that the index sequence $k_1, \ldots, k_\ell$ is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. Then

$$w_{k_1} \ldots w_{k_\ell} = z_{k_1} \ldots z_{k_\ell}. \tag{20}$$

Furthermore, we have the IO-reduction sequences

$f(k_1 \ldots k_\ell\#, \overline{k_1} \ldots \overline{k_\ell}\#, w_{k_1} \ldots w_{k_\ell}\#, \overline{z_{k_1}} \ldots \overline{z_{k_\ell}}\#) \to_R$

$f(k_1 \dots k_{\ell-1}\overline{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(\overline{k_1} \dots \overline{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(\overline{k_1} \dots \overline{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_{\ell-1}}\overline{w_{k_\ell}}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(\overline{k_1} \dots \overline{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, \overline{w_{k_1}} \dots \overline{w_{k_\ell}}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R$
$g(\overline{k_1} \dots \overline{k_\ell}\#, \overline{w_{k_1}} \dots \overline{w_{k_\ell}}\#)$

and

$f(k_1 \dots k_\ell\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R$
$f(k_1 \dots k_{\ell-1}w_{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(w_{k_1} \dots w_{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(w_{k_1} \dots w_{k_\ell}\#, \overline{k_1} \dots \overline{k_{\ell-1}z_{k_\ell}}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R^*$
$f(w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}}, \dots, \overline{z_{k_\ell}}\#, w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#) \to_R$
$h(w_{k_1} \dots w_{k_\ell}\#, \overline{z_{k_1}}, \dots, \overline{z_{k_\ell}}\#) = h(w_{k_1} \dots w_{k_\ell}\#, \overline{w}_{k_1} \dots \overline{w}_{k_\ell}\#)$.

Hence $g(\overline{k_1} \dots \overline{k_\ell}\#, \overline{w_{k_1}} \dots \overline{w_{k_\ell}}\#) \in L$ and $h(w_{k_1} \dots w_{k_\ell}\#, \overline{w}_{k_1} \dots \overline{w}_{k_\ell}\#) \in M$ have a common ancestor.

Conversely, assume that $p \in T_\Sigma(X)$ is a common ancestor of $g(\overline{k_1} \dots \overline{k_\ell}\#, q) \in L$ with $1 \le k_1, \dots, k_\ell \le n$, $\ell \ge 1$,

$$q \in T_{\overline{\Delta} \cup \{\#\}} \tag{21}$$

and $h(s,t) \in M$ with

$$s \in T_{\Delta \cup \{\#\}} \tag{22}$$

and

$$t \in T_{\overline{\Delta} \cup \{\#\}}. \tag{23}$$

Then p is a ground term. Hence

$$p = f(p_1, p_2, p_3, p_4) \text{ for some } p_1, p_2, p_3, p_4 \in T_\Sigma.$$

Furthermore, there are IO one-pass reduction sequences

$f(p_1, p_2, p_3, p_4) \to_R^* f(\overline{k_1} \dots \overline{k_\ell}\#, \overline{k_1} \dots \overline{k_\ell}\#, q, q) \to_R g(\overline{k_1} \dots \overline{k_\ell}\#, q)$,

and

$f(p_1, p_2, p_3, p_4) \to_R^* f(s,t,s,t) \to_R h(s,t)$.

Here

(a) $p_1 \Rightarrow_{R,IO} \overline{k_1} \dots \overline{k_\ell}\#$, (b) $p_1 \Rightarrow_{R,IO} s$,
(c) $p_2 \Rightarrow_{R,IO} \overline{k_1} \dots \overline{k_\ell}\#$, (d) $p_2 \Rightarrow_{R,IO} t$,
(e) $p_3 \Rightarrow_{R,IO} q$ and $p_3 \Rightarrow_{R,IO} s$, and (f) $p_4 \Rightarrow_{R,IO} q$ and $p_4 \Rightarrow_{R,IO} t$.

Consequently, (g) $p_1 \in T_{\Gamma \cup \{\#\}}$, $p_2 \in T_{\overline{\Gamma} \cup \{\#\}}$, $p_3 \in T_{\Delta \cup \{\#\}}$.

We proceed as follows.

(h) $p_1 = k_1 \dots k_\ell\#$ by (a) and (g).
(i) $p_2 = \overline{k_1} \dots \overline{k_\ell}\#$ by (c) and (g).
(j) $s = w_{k_1} \dots w_{k_\ell}\#$ by (22), (b), and (h).
(k) $t = \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#$ by (23), (d), and (i).
(l) $q = \overline{s}$ by (21), (22), (e), and (g).
(m) $q = t$ by (21), (23), and (f).
(n) $\overline{s} = t$ by (l) and (m).

By (j), (k), and (n),

$$\overline{w_{k_1} \dots w_{k_\ell}\#} = \overline{z_{k_1}} \dots \overline{z_{k_\ell}}\#.$$

Thus (20) holds. Hence the index sequence $k_1, \ldots, k_\ell$ is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. □

Theorem 3. *For right-linear TRSs, the second order common IO ancestor problem is undecidable.*

Proof. Let $\langle \mathbf{w}, \mathbf{z} \rangle$ be a Post Correspondence System over the alphabet Δ. We associated the ranked alphabet Σ, the recognizable tree languages L and M over Σ, and the TRS R over Σ with the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. By Lemma 9, the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ has a solution if and only if L and M have a common IO one-pass reduction ancestor. By Proposition 2, there is no algorithm which takes a PCS $\langle \mathbf{w}, \mathbf{z} \rangle$ as input and determines whether or not there is a solution of the PCS $\langle \mathbf{w}, \mathbf{z} \rangle$. □

6 Conclusion

We summed up the existing results in the literature and our contribution in Table 1.

We conjecture that for right-linear TRSs, the second order IO reachability problem and the second order common OI ancestor problem are decidable. We raise the following open problems.

Problem 1. *Given a TRS R and a recognizable tree language L, is it decidable whether $SFIO(L)$ is recognizable and whether $SFOI(L)$ is recognizable?*

For any linear TRS R, $\Rightarrow_{R,IO}$ is equal to $\Rightarrow_{R,OI}$. Hence for any linear TRS R and recognizable tree language L, $SFIO(L) = SFOI(L)$. Therefore, we raise the following question.

Problem 2. *Given a TRS R and a recognizable tree language L, is it decidable whether $SFIO(L) \subseteq SFOI(L)$, whether $SFOI(L) \subseteq SFIO(L)$, and whether $SFIO(L) = SFOI(L)$?*

References

[1] Baader, F. and Nipkow, T. Term Rewriting and All That, Cambridge University Press, Cambridge, United Kingdom, 1998.

[2] Van Den Brand, M., Klint, P., and Vinju, J. J. Term rewriting with traversal functions. ACM Transactions on Software Engineering and Methodology 12(2): 152-190 (2003).

[3] Bravenboer, M., Kalleberg, K. T., Vermaas R., and Visser E. Stratego/XT 0.17. A language and toolset for program transformation. Sci. Comput. Program. 72(1-2): 52-70 (2008).

[4] Dauchet, M., De Comité, F. A Gap Between Linear and Non Linear Term-Rewriting Systems. In Lescanne, P editor, Rewriting Techniques and Applications, RTA-87, Proceedings. Lecture Notes in Computer Science 256 Springer 1987, pages 95-104.

[5] Fülöp, Z., Jurvanen, E., Steinby, M., and Vágvölgyi, S. On one-pass term rewriting. In Brim, L., Gruska, J., Zlatuska, J. editors, Mathematical Foundations of Computer Science, 1998 Lecture Notes in Computer Science, **1450**, Springer Publishing Company, Berlin, 1998, pages 248-256; see also in Acta Cybernetica, **14** (1999) 83-98.

[6] Fülöp, Z. and Vágvölgyi, S. A Characterization of Irreducible Sets Modulo Left-Linear Term Rewriting Systems by Tree Automata, Fundamenta Informaticae, XIII (1990) 211-226.

[7] Gécseg, F. and Steinby, M. *Tree Automata* (Akadémiai Kiadó, Budapest, 1984).

[8] Gécseg, F. and Steinby, M. Tree languages. In G. Rozenberg and A. Salomaa, editors, Beyond Words, volume 3 of Handbook of Formal Languages, chapter 1, pages 1-68. Springer-Verlag, Berlin, 1997.

[9] Genet, T., Reachability analysis of rewriting for software verification, Habilitation document, 2009.
http://www.irisa.fr/celtique/genet/Publications/habilitation.pdf.

[10] Gilleron, R., Tison, S., Regular Tree Languages and Rewrite Systems. Fundamenta Informaticae **24** (1995) 157-174.

[11] Harrison, M. Introduction to Formal Language Theory, Addison-Wesley Publishing Company, Boston, MA, USA 1978.

[12] Seynhaeve, F., Tison, S., Tommasi, M., Homomorphisms and Concurrent Term Rewriting. In Ciobanu, G., Paun, G. editors Fundamentals of Computation Theory, 12th International Symposium, FCT '99, Iasi, Romania, 1999, Proceedings. Springer 1999 Lecture Notes in Computer Science **1684** (1999) pages 475-487.

[13] J. Ouaknine, J., Potapov I., Worrell, J., Reachability Problems - 8th International Workshop, RP 2014, Oxford, UK, September 22-24, 2014. Proceedings. Lecture Notes in Computer Science 8762, Springer 2014, ISBN 978-3-319-11438-5.

[14] Stratego/XT Manual

http://releases.strategoxt.org/strategoxt-manual/unstable/manual/one-page/index.html#tutorial-introduction.

[15] Visser, E. Program Transformation with Stratego/XT: Rules, Strategies, Tools, and Systems in Stratego/XT 0.9. In C. Lengauer, D. S. Batory, C. Consel, Odersky, M. editor, Domain-Specific Program Generation, International Seminar, Dagstuhl Castle, Germany, March 23-28, 2003, Revised Papers, Lecture Notes in Computer Science 3016 pages 216-238 Springer 2004.

Received 5th March 2014

Acta Cybernetica **22** (2016) 657–672.

On the Projection onto a Finitely Generated Cone

Miklós Ujvári*

Abstract

In the paper we study the properties of the projection onto a finitely generated cone. We show that this map is made up of finitely many linear parts with a structure resembling the facial structure of the finitely generated cone. An economical (regarding storage) algorithm is also presented for calculating the projection of a fixed vector, based on Lemke's algorithm to solve a linear complementarity problem. Some remarks on the conical inverse (a generalization of the Moore-Penrose generalized inverse) conclude the paper.

Keywords: projection map, finitely generated cone

1 Introduction

A standard way to generalize concepts in convex analysis is to replace subspaces with polyhedral cones, polyhedral cones with closed convex cones, closed convex cones with closed convex sets, and closed convex sets with closed convex functions. In this paper we make the first step on this way in the case of the concept of the projection onto a subspace, and examine the properties of the projection onto a finitely generated cone. (For higher levels of generality and applications – such as positive linear approximation problems and robotics –, see [3], [5], [6], [12], [13]. For recent results on the projection problem and related algorithms, see [2], [4].)

Let A be an m by n real matrix. Let $\operatorname{Im} A$ resp. $\operatorname{Ker} A$ denote the range space (that is the image space) and the null space (that is the kernel) of the matrix A. It is well-known that $(\operatorname{Im} A)^{\perp} = \operatorname{Ker}(A^T)$ and $(\operatorname{Ker} A)^{\perp} = \operatorname{Im}(A^T)$ where ^{T} denotes transpose and $^{\perp}$ denotes orthogonal complement (see [11]).

The projection map $p_{\operatorname{Im} A}$ onto the subspace $\operatorname{Im} A$ can be defined as follows: for every vector $y \in \mathcal{R}^m$, $p_{\operatorname{Im} A}(y)$ is the unique vector $Ax_* \in \mathcal{R}^m$ such that

$$||y - Ax_*|| = \min_{x \in \mathcal{R}^n} ||y - Ax||,\ x_* \in \mathcal{R}^n.$$

It is well-known that $p_{\operatorname{Im} A} : \mathcal{R}^m \to \mathcal{R}^m$ is a linear map: there exists a unique matrix $P_{\operatorname{Im} A} \in \mathcal{R}^{m \times m}$ such that

$$p_{\operatorname{Im} A}(y) = P_{\operatorname{Im} A} y\ (y \in \mathcal{R}^m).$$

*H-2600 Vác, Szent János utca 1., Hungary. E-mail: ujvarim@cs.elte.hu

DOI: 10.14232/actacyb.22.3.2016.7

The matrix $P_{\operatorname{Im} A}$ is symmetric (that is $P_{\operatorname{Im} A}^T = P_{\operatorname{Im} A}$), idempotent (that is $P_{\operatorname{Im} A}^2 = P_{\operatorname{Im} A}$), consequently positive semidefinite (that is $y^T P_{\operatorname{Im} A} y \geq 0$ for every $y \in \mathcal{R}^m$). Also the equalities

$$P_{\operatorname{Im} A} \cdot P_{\operatorname{Im} B} = 0,\ P_{\operatorname{Im} A} + P_{\operatorname{Im} B} = I$$

hold for any matrix B such that $\operatorname{Ker}(A^T) = \operatorname{Im} B$ (see [11]). (Here I denotes the identity matrix.)

Analogous results hold also in the case of the projection onto a finitely generated cone. Before stating the corresponding theorem we fix some further notation.

Let $\operatorname{Im}_+ A$ resp. $\operatorname{Ker}_+ A$ denote the so-called finitely generated cone

$$\operatorname{Im}_+ A := \{Ax \in \mathcal{R}^m : 0 \leq x \in \mathcal{R}^n\}$$

and the polyhedral cone

$$\operatorname{Ker}_+ A := \{x \in \mathcal{R}^n : Ax \geq 0\}.$$

A reformulation of the Farkas' lemma (see [9], [12] or [13]) claims that $(\operatorname{Im}_+ A)^* = \operatorname{Ker}_+(A^T)$ and $(\operatorname{Ker}_+ A)^* = \operatorname{Im}_+(A^T)$, where K^* denotes dual cone (or positive polar) of K, that is

$$K^* := \left\{a \in \mathcal{R}^d : a^T z \geq 0\ (z \in K)\right\},$$

for a convex cone $K \subseteq \mathcal{R}^d$. Thus polyhedral cones are the duals of the finitely generated cones, and vice versa. Furthermore, by the Farkas-Weyl-Minkowski theorem (see [9], [12] or [13]), for every matrix A there exists a matrix B such that $\operatorname{Im}_+ A = \operatorname{Ker}_+(B^T)$, or, dually, $\operatorname{Ker}_+(A^T) = \operatorname{Im}_+ B$. In other words, the finitely generated cones are polyhedral cones, and vice versa.

The projection map onto $\operatorname{Im}_+ A$ can be defined similarly as in the case of $\operatorname{Im} A$: for $y \in \mathcal{R}^m$ let $p_{\operatorname{Im}_+ A}(y)$ be the unique vector $Ax_* \in \mathcal{R}^m$ such that

$$||y - Ax_*|| = \min_{0 \leq x \in \mathcal{R}^n} ||y - Ax||,\ 0 \leq x_* \in \mathcal{R}^n.$$

For finitely generated cones which are not subspaces this map is not linear anymore, but is made up of linear parts, and these parts have the properties described already in the case of the projection onto a subspace. To state these facts precisely, the definitions of the faces, complementary faces and polyhedral partition are needed.

Let C be a convex set in $\mathcal{R}^d$. A convex set $F \subseteq C$ is called an *extremal subset* (or shortly a *face*) of the set C, if F does not contain an inner point of a line segment from C without containing the endpoints of the line segment, that is

$$c_1, c_2 \in C,\ 0 < \varepsilon < 1,\ \varepsilon c_1 + (1-\varepsilon)c_2 \in F \text{ implies } c_1, c_2 \in F.$$

We will denote by $F \lhd C$ the fact that F is a face of C.

If K is the finitely generated cone $\operatorname{Im}_+ A$, then its faces are finitely generated cones also, and there are only finitely many of them. The faces of (the finitely

generated cone) K^* are exactly the *complementary faces* $F^\triangle$ of the faces F of K, defined as

$$F^\triangle := F^\perp \cap (K^*) \ (F \lhd K).$$

The complementary face $F^\triangle$ can be alternatively defined as

$$F^\triangle = \{f_0\}^\perp \cap (K^*)$$

where f_0 is an arbitrary element of $\operatorname{ri} F$, the relative interior of the face F. Also $(F^\triangle)^\triangle = F$ holds for every face F of K (see [10] or [13], Theorem 7.27).

It is not difficult to verify using a standard separation argument that if $\mathcal{R}^d$ is the finite union of closed convex sets C_i with nonempty and pairwise disjoint interiors then the sets C_i are necessarily polyhedrons. In this case we call the set $\{C_i\}$ a *polyhedral partition* of $\mathcal{R}^d$.

Now, we can state our main result,

Theorem 1.1. *Let A be an m by n real matrix. Let K denote the finitely generated cone* $\operatorname{Im}_+ A$. *Then, the following statements hold:*

a) The cones $\{F - F^\triangle : F \lhd K\}$ form a polyhedral partition of $\mathcal{R}^m$. The map p_K is linear restricted to the members of this partition, that is for every $F \lhd K$ there exists a unique matrix $P_F \in \mathcal{R}^{m\times m}$ such that

$$p_K(f - g) = P_F \cdot (f - g) \ (f \in F, g \in F^\triangle).$$

b) For every $F \lhd K$, and every basis B of F, $P_F = P_{\operatorname{Im} B}$. Specially, the matrices P_F $(F \lhd K)$ are symmetric, idempotent, and positive semidefinite.

c) The map $P_.$ is a bijection between the sets $\{F : F \lhd K\}$ and $\{P_F : F \lhd K\}$; and it preserves the usual partial ordering on these sets, that is $F_1 \subseteq F_2$ if and only if $P_{F_2} - P_{F_1}$ is positive semidefinite.

d) For every face F of K,

$$P_F \cdot P^*_{F^\triangle} = 0, \ P_F + P^*_{F^\triangle} = I.$$

(Here $P^*_{F^\triangle}$ denotes the matrices defined by p_{K^*} obtained via replacing K with K^*, and $F \lhd K$ with $F^\triangle \lhd K^*$ in statement a).)

In Section 2 we will prove Theorem 1.1. In Section 3 we describe an algorithm for calculating the projection $p_{\operatorname{Im}_+ A}(y)$ for a fixed vector $y \in \mathcal{R}^m$. The method is based on Lemke's algorithm to solve a linear complementarity problem LCP (see [1]). After writing the problem as an LCP, using the structure of the problem our algorithm calculates with $r(A)$ by $2r(A)$ matrices instead of n by $2n$ matrices ($r(A)$ denotes rank of the matrix A). Finally, in Section 4 we describe a concept closely related to the projection $p_{\operatorname{Im}_+ A}$: the conical inverse $A^<$. Theoretical and algorithmical properties of the conical inverse are largely unexplored and need further research.

2 Proof of the main theorem

In this section we will prove Theorem 1.1. First we state three lemmas and propositions that will be used in the proof of statement a) in Theorem 1.1.

The first lemma describes a well-known characterization of the projection of a vector onto a closed convex cone, now specialized to the case of a finitely generated cone $K = \mathrm{Im}_+ A$, $A \in \mathcal{R}^{m\times n}$ (see [3], Proposition 3.2.3).

Lemma 2.1. *For every vector $y \in \mathcal{R}^m$ there exists a unique vector $k \in \mathcal{R}^m$ such that*

$$k \in K,\ k - y \in K^*,\ k^T(k-y) = 0. \tag{1}$$

This vector k equals $p_K(y)$ then.

As an immediate consequence, we obtain

Proposition 2.1. *Let F be a face of K. Let C_F denote the set of vectors y such that $p_K(y) \in \mathrm{ri}\, F$. Then,*

$$C_F = (\mathrm{ri}\, F) - F^{\triangle} \tag{2}$$

holds.

Proof. Let C denote the set on the right hand side of (2). First, we will show that $C_F \subseteq C$. Let y be an element of C_F, and let k denote the vector $p_K(y)$. Then $k \in \mathrm{ri}\, F$; and, by (1), $k - y \in \{k\}^{\perp} \cap K^*$, that is $k - y \in F^{\triangle}$. We can see that $y = k - (k-y)$ is an element of C, and the inclusion $C_F \subseteq C$ is proved.

Conversely, if $k \in \mathrm{ri}\, F$ and $k - y \in F^{\triangle}$, then (1) holds, so $k = p_K(y)$, and we can see that $y \in C_F$. This way we have proved the inclusion $C \subseteq C_F$ as well. □

The closure of the set C_F defined in Proposition 2.1 is

$$\mathrm{cl}\, C_F = F - F^{\triangle}. \tag{3}$$

The next lemma states that this finitely generated cone is full-dimensional (or equivalently has nonempty interior).

Lemma 2.2. *The linear hull of the set $F - F^{\triangle}$ is $\mathcal{R}^m$, for every face F of K.*

Proof. Let B be a matrix such that $K = \mathrm{Ker}_+ (B^T)$. It is well-known (see [9] or [13], Theorem 7.3) that then there exists a partition (B_1, B_2) of the columns of B such that

$$\begin{aligned} F &= \{y : B_1^T y \geq 0,\ B_2^T y = 0\}, \\ \mathrm{lin}\, F &= \{y : B_2^T y = 0\}, \\ \mathrm{ri}\, F &= \{y : B_1^T y > 0,\ B_2^T y = 0\}. \end{aligned}$$

(Here lin denotes linear hull.) Let $f_0 \in \mathrm{ri}\, F$. Then $F^{\triangle} = \{f_0\}^{\perp} \cap K^*$, and the latter set can easily be seen to be equal to $\mathrm{Im}_+ B_2$. Thus the linear hull of $F^{\triangle}$ is $\mathrm{Im}\, B_2$, the orthogonal complement of the linear hull of F. The linear hull of the set $F - F^{\triangle}$, being the sum of these two subspaces, is $\mathcal{R}^m$. □

It is well-known that relative interiors of the faces of a convex set form a partition of the convex set (see [7], Theorem 18.2): they are pairwise disjoint, and their union is the whole convex set. From this observation easily follows that the sets C_F are pairwise disjoint, and their union is the whole space. Consequently their closures, the sets $\operatorname{cl} C_F$ $(F \lhd K)$, have pairwise disjoint interiors (as the interior of $\operatorname{cl} C_F$ equals the interior of the convex C_F), cover the whole space, and (by Lemma 2.2) are full-dimensional. We obtained a proof of

Proposition 2.2. *The sets $F - F^{\triangle}$ $(F \lhd K)$ form a polyhedral partition of $\mathcal{R}^m$.*

We call a set $C \subseteq \mathcal{R}^m$,

- *positively homogeneous* if $0 < \lambda \in \mathcal{R}$, $y \in C$ implies $\lambda y \in C$;
- *additive* if $y_1, y_2 \in C$ implies $y_1 + y_2 \in C$.

Similarly, we call a map $p : C \to \mathcal{R}^m$, defined on a positively homogeneous, additive set C,

- *positively homogeneous* if $0 < \lambda \in \mathcal{R}$, $y \in C$ implies $p(\lambda y) = \lambda p(y)$;
- *additive* if $y_1, y_2 \in C$ implies $p(y_1 + y_2) = p(y_1) + p(y_2)$.

Proposition 2.3. *The sets C_F defined in Proposition 2.1 are positively homogeneous, additive sets; and the map p_K restricted to C_F is a positively homogeneous, additive map.*

Proof. The first half of the statement follows from Proposition 2.1: the sets $(\operatorname{ri} F) - F^{\triangle}$ are obviously positively homogeneous, additive sets.

To prove the remaining part of the statement, let $y \in C_F$. Then by Proposition 2.1 there exist $f_0 \in \operatorname{ri} F$, $g \in F^{\triangle}$ such that $y = f_0 - g$. Also, for $0 < \lambda \in \mathcal{R}$, $\lambda y \in C_F$. Again, by Proposition 2.1 there exist $f_0(\lambda) \in \operatorname{ri} F$, $g(\lambda) \in F^{\triangle}$ such that $\lambda y = f_0(\lambda) - g(\lambda)$. Necessarily,

$$\lambda p_K(y) = \lambda f_0 = f_0(\lambda) = p_K(\lambda y),$$

and we have proved the positive homogeneity of the map p_K restricted to the set C_F. Additivity can be similarly verified, so the proof is finished. □

We can see that the sets C_F are full-dimensional, positively homogeneous, additive sets, and the map p_K restricted to the set C_F is a positively homogeneous, additive map. Such maps have a unique linear extension as the following lemma states.

Lemma 2.3. *Let C be a positively homogeneous, additive set in $\mathcal{R}^m$ such that the linear hull of the set C is the whole space $\mathcal{R}^m$. Let $p : C \to \mathcal{R}^m$ be a positively homogeneous, additive map. Then there exists a unique linear map $\ell : \mathcal{R}^m \to \mathcal{R}^m$ such that $\ell(y) = p(y)$ for every $y \in C$.*

Proof. Let us choose a basis $\{y_1, \ldots, y_m\}$ from the set C, and let us define the map ℓ as follows:

$$\ell\left(\sum_{i=1}^{m} \lambda_i y_i\right) := \sum_{i=1}^{m} \lambda_i p(y_i) \ (\lambda_1, \ldots, \lambda_m \in \mathcal{R}).$$

We will show that the restriction of this linear map ℓ to the set C is the map p. Let C_0 denote the set of the linear combinations of the vectors $y_1, \ldots, y_m$ with positive coefficients. Then C_0 is an open set, and $p(y) = \ell(y)$ for every $y \in C_0$. Fix $y_0 \in C_0$, and let y be an arbitrary element from the set C. Then there exists a constant $0 < \varepsilon < 1$ such that the vector $y_\varepsilon := \varepsilon y + (1 - \varepsilon) y_0$ is in the set C_0. By positive homogeneity and additivity of the map p,

$$p(y_\varepsilon) = \varepsilon p(y) + (1 - \varepsilon) p(y_0).$$

On the other hand, by linearity of the map ℓ,

$$\ell(y_\varepsilon) = \varepsilon \ell(y) + (1 - \varepsilon) \ell(y_0).$$

Here $\ell(y_\varepsilon) = p(y_\varepsilon)$ and $\ell(y_0) = p(y_0)$, so we have $\ell(y) = p(y)$; the map ℓ meets the requirements.

Finally, uniqueness of the map ℓ is trivial, as ℓ must have fixed values for the elements of the full-dimensional set C. □

Now, we can describe the proof of Theorem 1.1.

Proof of part a) in Theorem 1.1: By Proposition 2.3 and Lemma 2.3 existence and uniqueness of matrices P_F follow such that

$$p_K(y) = P_F y \ (y \in C_F).$$

It is well-known (see Proposition 3.1.3 in [3]), that the map p_K is continuous, so we have actually

$$p_K(y) = P_F y \ (y \in \operatorname{cl} C_F).$$

We have seen already (see Proposition 2.2) that the sets $\operatorname{cl} C_F = F - F^\Delta$ $(F \lhd K)$ form a polyhedral partition of $\mathcal{R}^m$, thus the proof of statement a) in Theorem 1.1 is complete.

Proof of part b) in Theorem 1.1: Let F be a face of the cone K. Let B be a basis of the face F, and let B^Δ be a basis of the complementary face F^Δ. Then every vector $y \in \mathcal{R}^m$ can be written in the form

$$y = Bv + B^\Delta w, \ v \in \mathcal{R}^{\dim F}, \ w \in \mathcal{R}^{\dim F^\Delta}.$$

Multiplying this equality from the left with the matrices P_F and B^T, respectively, we obtain the equalities

$$P_F y = Bv, \ B^T y = B^T B v.$$

These equalities imply

$$P_F y = Bv = B(B^T B)^{-1} B^T y = P_{\mathrm{Im}\, B} y.$$

We have $P_F = P_{\mathrm{Im}\, B}$, and the proof of statement b) in Theorem 1.1 is finished also.

Proof of part c) in Theorem 1.1: First, notice that the map P is trivially a bijection between the sets $\{F : F \lhd K\}$ and $\{P_F : F \lhd K\}$. (Injectivity follows from the obvious fact that if $F_1 \neq F_2$, for example there exists $y \in F_1 \setminus F_2$, then $P_{F_1} = P_{F_2}$ would imply $P_{F_2} y = P_{F_1} y = y$, and thus $y \in F_2$, which is a contradiction.)

Hence, we have to verify only that $F_1, F_2 \lhd K$, $F_1 \subseteq F_2$ implies that $P_{F_2} - P_{F_1}$ is positive semidefinite. Let B_1 be a basis of the face F_1, and let B_2 be a basis of the face F_2 such that $B_1 \subseteq B_2$. Then for every $y \in \mathcal{R}^m$, by the definition of the projection map,

$$||y - P_{\mathrm{Im}\, B_1} y||^2 \geq ||y - P_{\mathrm{Im}\, B_2} y||^2.$$

This inequality, by part b) in Theorem 1.1 implies that

$$y^T P_{F_2} y \geq y^T P_{F_1} y \ (y \in \mathcal{R}^m),$$

that is the positive semidefiniteness of the matrix $P_{F_2} - P_{F_1}$, which was to be shown.

Proof of part d) in Theorem 1.1: Let $y_1, \ldots, y_m$ be a basis in the set C_F, and let $y_1^\triangle, \ldots, y_m^\triangle$ be a basis in the set $C^*_{F^\triangle}$.

Then, to prove that $P_F \cdot P^*_{F^\triangle} = 0$, it is enough to show that

$$y_i^T P_F \cdot P^*_{F^\triangle} y_j^\triangle = 0 \ (1 \leq i, j \leq m).$$

In other words we have to show that the vectors $p_K(y_i)$ and $p_{K^*}(y_j^\triangle)$ are orthogonal. This follows from the fact that the former vectors are in F, while the latter vectors are in $F^\triangle$.

To prove the equality $P_F + P^*_{F^\triangle} = I$, it is enough to verify that

$$y_i^T (P_F + P^*_{F^\triangle}) y_j^\triangle = y_i^T y_j^\triangle \ (1 \leq i, j \leq m).$$

In other words that

$$p_K(y_i)^T y_j^\triangle + y_i^T p_{K^*}(y_j^\triangle) = y_i^T y_j^\triangle \ (1 \leq i, j \leq m),$$

or equivalently that

$$(y_i - p_K(y_i))^T (y_j^\triangle - p_{K^*}(y_j^\triangle)) = 0 \ (1 \leq i, j \leq m).$$

This latter equality is the consequence of the fact that the vectors $p_K(y_i) - y_i$ are in $F^\triangle$ while the vectors $p_{K^*}(y_j^\triangle) - y_j^\triangle$ are in $(F^\triangle)^\triangle = F$ and so are orthogonal.

This way we have finished the proof of part d), and the proof of Theorem 1.1 as well. □

We conclude this section with an illustrative example. Let us consider the following vectors: $a_1 := (1,0)$ and $a_2 := (1,1)$, and let $K := \operatorname{cone}\{a_1, a_2\}$ (cone denotes convex conical hull). Let K^* denote the dual cone of K. Then, K^* can be described as $K^* = \operatorname{cone}\{a_1^\perp, a_2^\perp\}$, with $a_1^\perp = (0,1)$ and $a_2^\perp = (1,-1)$.

With these notations the faces of K and K^* can be given as follows:

$$F_0 = \{0\},\ F_1 = \operatorname{cone}\{a_1\},\ F_2 = \operatorname{cone}\{a_2\},\ F_3 = \operatorname{cone}\{a_1, a_2\} = K,$$
$$F_0^\triangle = \operatorname{cone}\{a_1^\perp, a_2^\perp\} = K^*,\ F_1^\triangle = \operatorname{cone}\{a_1^\perp\},\ F_2^\triangle = \operatorname{cone}\{a_2^\perp\},\ F_3^\triangle = \{0\}.$$

By the help of these faces we obtain the following polyhedral partition of $\mathcal{R}^2$:

$$C_0 = F_0 - F_0^\triangle = -K^*,\ C_1 = F_1 - F_1^\triangle = \operatorname{cone}\{a_1, -a_1^\perp\}$$
$$C_2 = F_2 - F_2^\triangle = \operatorname{cone}\{a_2, -a_2^\perp\},\ C_3 = K.$$

The projection onto the cone K can be described as follows: for $y = (y_1, y_2)$,

$$p_K(y) = \begin{cases} 0, & \text{if } y \in C_0 = -K^*, \\ (y_1, 0), & \text{if } y \in C_1, \\ (y_1 + y_2, y_1 + y_2)/2, & \text{if } y \in C_2, \\ y, & \text{if } y \in C_3 = K. \end{cases}$$

3 Algorithm for computing the projection

In this section we will describe an algorithm for calculating the projection of a fixed vector onto a finitely generated cone. The algorithm economically solves a certain type of linear complementarity problems as well.

Let A be an m by n real matrix, and let K denote the finitely generated cone $\operatorname{Im}_+ A$. Let us fix a vector $y \in \mathcal{R}^m$. To compute the projection $p_K(y)$, by Lemma 2.1, we have to find a vector $x \in \mathcal{R}^n$ such that

$$x \geq 0,\ Ax - y \in \operatorname{Ker}_+(A^T),\ (Ax)^T(Ax - y) = 0.$$

This problem can be rewritten as a linear complementarity problem

$$LCP(A): \quad \begin{cases} \text{Find vectors } z, x \in \mathcal{R}^n \text{ such that} \\ z - A^TAx = -A^Ty;\ x, z \geq 0;\ z^Tx = 0. \end{cases}$$

A finite version of Lemke's algorithm (see [1]) can be applied to solve $LCP(A)$; if (x, z) is a solution, then $Ax = p_K(y)$ is the projection we searched for.

However, a more economical algorithm can be constructed to find the projection of a vector; economical in the sense that instead of solving the n-dimensional $LCP(A)$, it solves a sequence of $r(A)$-dimensional LCPs, $LCP(B_1), LCP(B_2), \ldots$, where $B_1, B_2, \ldots$ are bases of the matrix A. (A matrix B is called a *basis* of the matrix A, if B is an m by $r(A)$ submatrix of A, and $r(B) = r(A)$.)

Before describing this algorithm, we prove three propositions and lemmas that will be used in the proof of the correctness and finiteness of the algorithm.

Let B be a basis of the matrix A, with corresponding basis tableau $T(B)$. (The *basis tableau* $T(B)$ corresponding to a basis B contains the unique coefficients $t_{ij} \in \mathcal{R}$ such that

$$a_j = \sum_i \{t_{ij} a_i : a_i \in B\} \ (a_j \in A),$$

where a_j denotes the j-th column vector of the matrix A.) We use the basis tableau $T(B)$ for notational convenience only. In each step of the algorithm we will use only the j_*-th column of the tableau, that is the solution t of the system $Bt = a_{j_*}$. Note that t can be calculated using $r(A)$ by $r(A)$ matrices: it is enough to find an $r(A)$ by $r(A)$ invertible submatrix of B, and solve the corresponding subsystem of $Bt = a_{j_*}$.

With these notations,

Proposition 3.1. *If (x, z) is a solution of LCP(B), then $Bx = p_{\mathrm{Im}_+ B}(y)$. Furthermore, if*

$$a_j^T(Bx - y) \geq 0 \ (a_j \in A \setminus B) \tag{4}$$

holds, then $Bx = p_{\mathrm{Im}_+ A}(y)$ as well.

Proof. The statement follows from Lemma 2.1, see also the remark made at the beginning of this section. □

If (4) does not hold, then there exist column vectors $a_j \in A \setminus B$ such that $a_j^T(Bx - y) < 0$. Choose one of them which minimizes the inner product with the vector $Bx - y$: let j_* be an index such that

$$a_{j_*}^T(Bx - y) = \min \left\{ a_j^T(Bx - y) : a_j \in A \setminus B \right\}, \ a_{j_*} \in A \setminus B. \tag{5}$$

This vector a_{j_*} will enter the basis B.

From the definition of the index j_* immediately follows

Lemma 3.1. *The minimum in (5) is less than* 0. □

Now, we will choose the vector a_{i^*} that leaves the basis B. Let i^* denote an index such that

$$a_{i^*}^T(Bx - y) = \max \left\{ a_i^T(Bx - y) : a_i \in B, t_{ij_*} \neq 0 \right\}, \ a_{i^*} \in B, \ t_{i^* j_*} \neq 0. \tag{6}$$

Remember that $Bx = p_{\mathrm{Im}_+ B}(y)$; so $a_i^T(Bx - y) \geq 0$ holds for every vector $a_i \in B$. Hence the maximum in (6) is at least 0. But more can be claimed:

Lemma 3.2. *The maximum in (6) is greater than* 0.

Proof. The vector a_{j_*} can be written in the form

$$a_{j_*} = \sum \{t_{ij_*} a_i : a_i \in B\}.$$

If $a_i^T(Bx - y) = 0$ would hold for every index i such that $a_i \in B$, $t_{ij_*} \neq 0$, then $a_{j_*}^T(Bx - y) = 0$ would follow, contradicting Lemma 3.1. □

It is well-known (see [11]) that $t_{i^*j_*} \neq 0$ implies

Proposition 3.2. *The submatrix*

$$\hat{B} := (B \setminus \{a_{i^*}\}) \cup \{a_{j_*}\}$$

is a basis of the matrix A. Furthermore, the corresponding basis tableau $T(\hat{B})$ can be obtained from the basis tableau $T(B)$ by pivoting on the (i^*, j_*)-th position of the latter tableau.

For the new basis $\hat{B}$ it holds that

Lemma 3.3. *The vectors $p_{\mathrm{Im}_+ B}(y)$ and a_{j_*} are elements of the cone $\mathrm{Im}_+ \hat{B}$.*

Proof. As for the solution (x, z) of $LCP(B)$, $z^T x = 0$, $z, x \geq 0$ holds, necessarily $x_i z_i = 0$ for all indices i. As by Lemma 3.2 $z_{i^*} = a_{i^*}^T(Bx - y) > 0$, we have $x_{i^*} = 0$. We can see that $Bx \in \mathrm{Im}_+ (B \setminus \{a_{i^*}\})$; consequently $Bx \in \mathrm{Im}_+ \hat{B}$, that is $p_{\mathrm{Im}_+ B}(y) \in \mathrm{Im}_+ \hat{B}$. The remaining statement that $a_{j_*} \in \mathrm{Im}_+ \hat{B}$ is trivial. □

The next proposition shows that the new basis $\hat{B}$ is an improvement over B.

Proposition 3.3. *The distance between $p_{\mathrm{Im}_+ \hat{B}}(y)$ and y is less than the distance between $p_{\mathrm{Im}_+ B}(y)$ and y.*

Proof. Let S denote the following set of vectors:

$$S := \{s = \varepsilon Bx + (1 - \varepsilon)a_{j_*} : 0 < \varepsilon < 1, \varepsilon \in \mathcal{R}\}.$$

By Lemma 3.3, $S \subseteq \mathrm{Im}_+ \hat{B}$. Furthermore, from Lemma 3.1 it can easily be seen that there exists a vector $s \in S$ such that the distance between y and s is less than the distance between y and $Bx = p_{\mathrm{Im}_+ B}(y)$. (In fact, it can easily be verified that

$$\frac{||y - s||^2 - ||y - Bx||^2}{2(1 - \varepsilon)} \to a_{j_*}^T(Bx - y) \ (\varepsilon \to 1).$$

Applying Lemma 3.1, we have $||y - s|| < ||y - Bx||$ for some $s \in S$.) As the distance between y and $p_{\mathrm{Im}_+ \hat{B}}(y)$ is not greater than the distance between y and s, so the statement follows. □

Now, we can describe the algorithm and prove its correctness.

Theorem 3.1. *Algorithm 1 finds $p_K(y)$ after finite number of steps.*

Proof. The correctness of the algorithm follows from Propositions 3.1 and 3.2. The finiteness of the algorithm is a consequence of Proposition 3.3: as the distance between the vectors $p_{\mathrm{Im}_+ B}(y)$ and y decreases with each step, so there can be no repetition in the sequence of the bases, and there are only finitely many of the bases of A. □

Algorithm 1 Algorithm for computing the projection economically.

Funct PROJ(y, A)

Let B be any basis of A, and set $e := 0$
while ($e = 0$) **do**
  Compute a solution (x, z) of $LCP(B)$
  if (4) holds **then**
    Set $e := 1$
  else
    Choose j_* according to (5)
    Choose i^* according to (6)
    Set $B := B \cup \{a_{j_*}\} \setminus \{a_{i^*}\}$
  end if
end while
return Bx

Finally, we remark that Algorithm 1 can be applied to solve economically any LCP of the form

$$(LCP): \text{ Find } x, z \text{ such that } z - Mx = q;\ z, x \geq 0;\ z^T x = 0,$$

where M is a symmetric positive semidefinite matrix, $q \in \operatorname{Im} M$. To see this it is enough to note that a matrix A can be found such that $M = A^T A$ using Cholesky decomposition (see [11]). Then $\operatorname{Im} M = \operatorname{Im}(A^T)$, so $q = -A^T y$ for some vector y. This way we have rewritten (LCP) as $LCP(A)$, and the algorithm discussed in this section can be applied to solve the problem.

4 Remarks on the conical inverse

In this section we describe a concept closely related to the projection onto a finitely generated cone and also to the Moore-Penrose generalized inverse of a matrix.

Let A be an m by n real matrix. For every vector $y \in \mathcal{R}^m$ there exists a unique vector $x_* \in \mathcal{R}^n$ such that $Ax_* = p_{\operatorname{Im} A}(y)$ and

$$||x_*|| = \min\{||x|| : Ax = p_{\operatorname{Im} A}(y),\ x \in \mathcal{R}^n\}.$$

The dependence of the vector x_* on the vector y turns out to be linear (see [11]): there exists a unique matrix, called the *Moore-Penrose generalized inverse* of the matrix A and denoted by A^- such that $x_* = A^- y$ for every $y \in \mathcal{R}^m$.

It is also well-known (see [8]) that the Moore-Penrose generalized inverse can be alternatively defined as the unique matrix $A^- \in \mathcal{R}^{n \times m}$ satisfying the four conditions: a) $AA^-A = A$; b) $A^-AA^- = A^-$; c) $(AA^-)^T = AA^-$; d) $(A^-A)^T = A^-A$.

Similarly as in the case of the projection map, this concept can also be generalized via replacing the subspace $\operatorname{Im} A$ with the finitely generated cone $\operatorname{Im}_+ A$. The map defined this way is called the *conical inverse* of the matrix A, and is denoted

by $A^<$. Thus for every vector $y \in \mathcal{R}^m$, $A^<(y)$ is the unique vector in $\mathcal{R}^n$ satisfying $AA^<(y) = p_{\mathrm{Im}_+ A}(y)$, $A^<(y) \geq 0$, and

$$||A^<(y)|| = \min\{||x|| : Ax = p_{\mathrm{Im}_+ A}(y),\ x \geq 0\}.$$

The next proposition describes a certain relation between the two inverses defined above.

Proposition 4.1. *For every vector $y \in \mathcal{R}^m$, it holds that*

$$(A, -A)^<(y) = (\max\{A^- y, 0\}, -\min\{A^- y, 0\})$$

where the max and min are meant elementwise.

Proof. Let us consider the following two programs:

$$(\hat{P}): \qquad \text{Find} \min_{x_1, x_2 \geq 0} ||y - Ax_1 + Ax_2||$$

and

$$(P): \qquad \text{Find} \min_{x} ||y - Ax||.$$

The variable transformations

$$x := x_1 - x_2 \text{ resp. } x_1 := \max\{x, 0\},\ x_2 := -\min\{x, 0\}$$

show the equivalence of programs $(\hat{P})$ and (P).

Furthermore, it can easily be seen that

- if x is an optimal solution of program (P), then the vector

  $$(x_1, x_2) := (\max\{x, 0\} + p, -\min\{x, 0\} + p)$$

  is an optimal solution of program $(\hat{P})$ for any vector $p \geq 0$;
- if (x_1, x_2) is an optimal solution of program $(\hat{P})$, then the vector $x := x_1 - x_2$ is an optimal solution of program (P) such that

  $$(x_1, x_2) = (\max\{x, 0\} + p, -\min\{x, 0\} + p)$$

  for some vector $p \geq 0$.

Consequently, the optimal solution of $(\hat{P})$ with the least norm will be

$$(x_1, x_2) = (\max\{x, 0\}, -\min\{x, 0\}),$$

where x is the optimal solution of (P) with the least norm; which was to be shown. □

Thus any algorithm for calculating the conical inverse can be used for calculating the Moore-Penrose generalized inverse. Conversely also, as the following proposition shows.

Proposition 4.2. *The vector x equals $A^<(y)$ if and only if for some vector z, (x, z) is a solution of the following linear complementarity problem:*

$$(LCP^<): \quad \begin{cases} \text{Find } x, z \text{ such that } x, z \geq 0; z^T x = 0; \\ x - (I - A^- A)z = A^- p_{\mathrm{Im}_+ A}(y). \end{cases}$$

Proof. Let $x_0 \geq 0$ be a vector such that $Ax_0 = p_{\mathrm{Im}_+ A}(y)$. To find $A^<(y)$ we have to find the unique optimal solution of the following convex quadratic program:

$$(QP): \quad \text{Find} \quad \min\left\{\frac{1}{2}||x||^2 : Ax = Ax_0,\ x \geq 0\right\}.$$

By the Kuhn-Tucker conditions (see [1]), the vector $x \geq 0$ is an optimal solution of program (QP) if and only if there exists a vector $z \geq 0$ such that $z^T x = 0$,

$$x - z \in \mathrm{Im}\,(A^T),\ x - x_0 \in \mathrm{Ker}\,A. \tag{7}$$

It is well-known that

$$P_{\mathrm{Im}\,(A^T)} = A^- A,\ P_{\mathrm{Ker}\,A} = I - A^- A,$$

so (7) can be rewritten as

$$(I - A^- A)(x - z) = 0,\ A^- Ax = A^- Ax_0. \tag{8}$$

It is easy to see that (8) holds if and only if (x, z) satisfies the following equality

$$x - (I - A^- A)z = A^- Ax_0.$$

We can see that $x = A^<(y)$ if and only if there exists a vector z such that (x, z) is a solution of $(LCP^<)$; the proof is complete. □

Note that any problem of minimizing a strictly convex quadratic function

$$q(x) := \frac{1}{2}x^T Mx + c^T x\ (x \in \mathcal{R}^n)$$

(with $M = V^T V \in \mathcal{R}^{n\times n}$ symmetric positive definite, $V \in \mathcal{R}^{n\times n}$ invertible, $c \in \mathcal{R}^n$) over a non-empty polyhedron

$$\tilde{P} := \{x : \tilde{A}x = \tilde{b},\ x \geq 0\}$$

(with $\tilde{A} \in \mathcal{R}^{m\times n}$, $\tilde{b} \in \mathcal{R}^m$) can be transformed (applying an invertible affine transformation of $\mathcal{R}^n$, namely $x := Vx + V^{-1T}c$, $x \in \mathcal{R}^n$) into the form of (QP), and in turn (QP) can be formulated (homogenized) as a projection (of the vector $(0, 1) \in \mathcal{R}^{n+1}$) problem onto the polyhedral cone

$$\left\{\begin{pmatrix} x \\ \lambda \end{pmatrix} : Ax - \lambda \cdot Ax_0 = 0,\ x \geq 0,\ \lambda \geq 0\right\} \subseteq \mathcal{R}^{n+1}$$

(or the constraint $Ax = Ax_0$ can be replaced with $x = x_0 + B\tilde{x}$, where B is a basis of $\operatorname{Ker} A$, which results again in a projection problem), see [6]. Similarly, the minimizing of the strictly convex quadratic function $q(x)$ over a *finitely generated set* $\tilde{Q} + \tilde{R}$ (with

$$\begin{aligned} \tilde{Q} &:= \left\{ \sum_{i=1}^{k} \lambda_i \tilde{y}_i : \lambda_i \geq 0, \sum_{i=1}^{k} \lambda_i = 1 \right\}, \\ \tilde{R} &:= \left\{ \sum_{j=1}^{\ell} \mu_j \tilde{z}_j : \mu_j \geq 0 \right\} \end{aligned}$$

for some vectors $\tilde{y}_i, \tilde{z}_j \in \mathcal{R}^n$) can be reduced to solving a projection problem onto a finitely generated cone of the form

$$\left\{ \sum_{i=1}^{k} \lambda_i \begin{pmatrix} y_i \\ 1 \end{pmatrix} + \sum_{j=1}^{\ell} \mu_j \begin{pmatrix} z_j \\ 0 \end{pmatrix} : \lambda_i, \mu_j \geq 0 \right\}$$

for some vectors $y_i, z_j \in \mathcal{R}^n$. The proof of this statement is an adaptation of the results in Chapters 5 and 6 of [6] (as finitely generated sets are polyhedrons, see [7], Theorem 19.1, or [12], [13]), and is left to the reader.

Finally, we mention some open problems concerning the projection and the conical inverse:

- Testing Algorithm 1 on numerical examples (and the comparison of its efficiency with Lemke's algorithm) is a possible direction for further research. What is the number of the smaller LCPs we have to solve in the course of the algorithm?

- Is it true, that similarly to the case of the projection, the conical inverse is also continuous and made up from linear parts? (This statement is trivial if the m by n matrix A has rank $r(A) = n$.)

- We can see from Proposition 4.2 that the conical inverse for a fixed vector y can be calculated via solving an n-dimensional LCP. Is it possible to construct an algorithm to compute $A^<(y)$ more economically, similarly as in the case of the projection map? Can this algorithm (or a combination of the two algorithms) be used for solving economically general classes of LCPs?

5 Conclusion

In this paper we examined the properties of the projection onto a finitely generated cone. Our main result shows that this map is made up of linear parts with a structure resembling the facial structure of the finitely generated cone we project onto (the map is linear if and only if we project onto a subspace). Also we presented

an algorithm for computing the projection of a fixed vector. The algorithm is economical in the sense that it calculates with matrices whose size depends on the dimension of the finitely generated cone and not on the number of the generating vectors of the cone. Some remarks and open problems concerning the conical inverse conclude the paper.

Acknowledgments

I thank János Fülöp and one of the anonymous referees for calling my attention to reference [5], and [2], [4], respectively.

References

[1] Bazaraa, M. S. and Shetty, C. M. *Nonlinear Programming, Theory and Algorithms.* John Wiley & Sons, New York, 1979.

[2] Ekárt, A., Németh, A. B., and Németh, S. Z. Rapid heuristic projection on simplicial cones. Manuscript, 2010.

[3] Hiriart-Urruty, J. and Lemaréchal, C. *Convex Analysis and Minimization Algorithms I.* Springer-Verlag, Berlin, 1993.

[4] Hu, X. An exact algorithm for projection onto a polyhedral cone. *Australian & New Zealand Journal of Statistics*, 40: 165–170, 1998.

[5] Lawson, C. L. and Hanson R. J. *Solving Least Square Problems.* Prentice Hall, Englewood Cliffs NJ, 1974.

[6] Liu, Z. *The Nearest Point Problem in a Polyhedral Cone and its Extensions.* Phd thesis, North Carolina State University, 2009.

[7] Rockafellar, R. T. *Convex Analysis.* Princeton University Press, Princeton NJ, 1970.

[8] Rózsa, P. *Lineáris Algebra és Alkalmazásai.* Tankönyvkiadó, Budapest, 1991.

[9] Schrijver, A. *Theory of Linear and Integer Programming.* John Wiley & Sons, New York, 1986.

[10] Stoer, J. and Witzgall, C. *Convexity and Optimization in Finite Dimensions I.* Springer-Verlag, Berlin, 1970.

[11] Strang, G. *Linear Algebra and its Applications.* Academic Press, New York, 1980.

[12] Ujvári, M. *A Szemidefinit Programozás Alkalmazásai a Kombinatorikus Optimalizálásban.* ELTE Eötvös Kiadó, Budapest, 2001.

[13] Ujvári, M. *Konvex Analízis.* Manuscript, 2009. URL: http://www.oplab.sztaki.hu/tanszek/letoltes.htm

Received 12th August 2013

Acta Cybernetica **22** (2016) 673–685.

Robustness of BitTorrent-like VoD Protocols

Tamás Vinkó*

Abstract

Besides server supported solutions for Video-on-demand, approaches based on distributed systems such as BitTorrent are being used due to their efficiency and high scalability. There are several protocol variants proposed in the literature, which are mainly concerned with providing mechanisms for piece selection and peer selection. In this paper, using the concept of Design Space Analysis, we give comparisons of the performances of several BitTorrent-like Video-on-demand protocols under the assumption that other protocol variants may also enter the system.

Keywords: Design Space Analysis, BitTorrent, Video-on-demand, Piece selection, Peer selection

1 Introduction

Among the peer-to-peer (P2P) content sharing applications BitTorrent [6] is inevitably the most popular protocol. One of its key ideas is that a file, which is subject to be shared, is divided into small pieces and the participating peers are exchanging these pieces between each other. For traditional file sharing it is a good strategy to request the rarest first piece to be downloaded. That is indeed the default option in BitTorrent. However, in case of video-on-demand (VoD) systems, where users would like to watch or listen the media content (video or audio), the pieces of the file should be received in an in-order fashion and the system also needs some quality-of-service (QoS) requirements to be fulfilled, like smooth playback and quick startup. Contrasting this service with live streaming, different users in VoD systems are usually interested in different parts of the media content, which appears to be similar to the dynamics of users in traditional P2P file-sharing. Thus, recently, much attention has been paid to the VoD extensions of the BitTorrent protocol [5, 11, 13, 18, 19].

Two of the most important aspects of BitTorrent-like VoD approaches are piece selection and peer selection. As it has been shown by D'Acunto *et al.* [7], the proposed solutions to these two policies can lead to different performance and thus they should be chosen according to the given scenario. On the other hand, performance evaluations of the VoD protocols in a mixed swarm, where peers can use

*University of Szeged, Hungary, E-mail: tvinko@inf.u-szeged.hu

DOI: 10.14232/actacyb.22.3.2016.8

different protocol variants have not yet been considered in the literature. To this end, in this paper we apply the approach of Design Space Analysis [16] and investigate the ability of VoD protocol variants outperforming other variants in a mixed population with respect to the two aforementioned QoS requirement measures. In this way, besides the Performance, which indicates the average efficiency of a given protocol in a homogeneous population, the Robustness and Aggressiveness of the considered protocols can also be given.

In the following we first give the necessary definitions of the Design Space Analysis and the BitTorrent-like VoD systems, see Section 2. Then we set up the design space of the VoD protocols by giving all the details of its dimensions and measures, see Section 3. In Section 4 we describe the simulation model and methodology applied in the paper. Finally, in Section 5 the simulation results are presented.

2 Background

In this section all the important definitions used later in the paper are introduced.

Design Space Analysis. In order to comprehensively model incentives in distributed protocols, a method called Design Space Analysis (DSA) was proposed by Rahman *et al.* [16]. This simulation based approach was inspired by Axelrod [4] modeling specific interactions in repeated games. The DSA framework combines the specification of a design space of protocols together with their analysis by means of extensive simulations. The specification part consists two steps: parametrization and actualization. Parametrization is the determination of the salient dimensions of the design space. In actualization one specifies the actual values of every individual dimension. Having done with these two steps, a solution concept can be used, in which any protocol of the design space can be characterized using different measures. For a given protocol Performance, Robustness and Aggressiveness measures (called PRA quantification) were proposed [16]. Using a predefined utility, a protocol's Performance is the average performance of the system when all the participating peers apply the same protocol variant. This predefined utility, which quantifies individual performance, is always domain specific. For example, in traditional P2P based content distribution it can be the peers download speed. Robustness and Aggressiveness indicates the ability of outperforming other protocol variants depending on the composition of the system, e.g. the protocol is used by a majority, respectively a minority, of the population. Using these three measures, provided that they are normalized into the interval $[0, 1]$, the properties of a given protocol can be characterized as a point in the three dimensional PRA space $[0, 1]^3$. System designers might want to introduce protocols which maximize on all dimensions. However, usually one needs to compromise between the three. As it was shown in [16], this is indeed the case regarding a large space of P2P file swarming systems.

BitTorrent. The most popular P2P based content distribution protocol is BitTorrent. In this system, the files are divided into small pieces, which allows the participating peers to share the parts of the file which they already obtained while still downloading the other pieces. Peer discovery is usually done with the help of a central component (tracker) that, upon requests, sends a list of nodes participating in the downloading (leeching) and uploading (seeding) of the same content. Each peer maintains connections to its neighbours and exchanges the pieces with the subset of them. Technically, each peer equally divides its upload capacity into upload slots of two types: regular unchoke slots and optimistic unchoke slots. Regular unchoke slots are assigned to peers that have recently provided data directly to the peer at their highest speeds. The assignments of these slots are re-evaluated in every unchoke interval δ, usually $\delta = 10$ seconds. Optimistic unchoke slots, on the other hand, are assigned to randomly selected peers. This allocation is also re-evaluated in every 3δ interval. This scheme can be seen as a hill-climbing type optimization method of finding out the peers from which the file can be obtained at the highest possible speed. At the same time it also helps to newcomers starting their download despite the fact that they have nothing to share at the beginning.

Each peer informs (and gets informed by) its neighbours about the pieces it owns (they own). Using this information, the request of pieces are done by local rarest first policy, e.g. each peer requests pieces which are the rarest among its neighbours. The outcome of this policy is that less available pieces of the file get replicated among the peers.

3 Setting up the design space

In this section, the application of DSA for BitTorrent-like VoD systems is described. First, we Parameterize the generic design space giving its dimensions. Next, based on this space, we Actualize a specific BitTorrent-like VoD design space. This is then followed by the application of the PRA quantification on this space.

3.1 Parametrization

In our design space we specify two dimensions relevant to the VoD protocols:

- *Piece Selection* determines which piece of the video file should be selected for downloading by the peer. The role of this policy is to find a trade-off between in-order piece download (for QoS) and high enough bartering chances among peers (to guarantee efficient bandwidth utilization). An important parameter of this dimension is called *sequentiality parameter* to be tuned to favour one of these two aspects.

- *Peer selection* dictates how peers selects each other to upload data to. The role of this policy is to incentivize cooperation among peers. In BitTorrent-like systems it is usually designed to favour good uploaders.

3.2 Actualization

Having defined the two dimensions of our design space, the next step is to specify the actual values for every individual dimensions.

Three different policies were investigated for *Piece selection*:

- *Window-based* piece selection (WIN) employs a sliding window of fixed size within which pieces are chosen to download [15]. Depending on the sequential download progress of the peer the window advances from the start to the end of the media file. Within this sliding window BitTorrent's standard rarest-first piece selection is applied. The size of the window, denoted by w, is the sequentiality parameter and naturally has effect on the piece diversity among the peers.

- *Probabilistic* piece selection (PROB) chooses pieces in relation to a fixed probability distribution, usually giving higher chances to first pieces not yet downloaded. In our simulations we employed the Zipf probability distribution as proposed in [12], which has the form $P(k) \sim (k + 1 - k_0)^{-\theta}$, where k_0 is the index of the first piece that the peer has not yet downloaded. In this case the parameter θ represents the sequentiality parameter.

- *Priority-based* piece selection (PRIO) gives priority to pieces which are close to be played. This policy, as it was defined in [9], uses a parameter h which defines the size of the priority set (and thus it is the sequentiality parameter of this policy) in the following way. Assume that the peer's current playback position is t. The peer requests the piece k on the first match in the following list of sets of pieces (called priority sets):

  - high priority: $t \leq k < t + h$ in-order piece selection if the local peer has already started playback, rarest fist otherwise;
  - mid priority: $t + h \leq k < t + 5h$: with rarest first piece selection;
  - low priority: $t + 5h \leq k$: with rarest first piece selection.

In the dimension of *Peer selection* we studied the following three policies:

- *Direct reciprocity* (D), which is the standard peer selection policy of BitTorrent. Using this policy a peer uploads to other peers that have recently uploaded to it at the highest rate. This decision is taken based on local information only, that is, the peers themselves are measuring the upload speed of the other peers.

- *Indirect reciprocity* (I) prescribes that a peer uploads to other nodes that have recently forwarded data to others at the highest rate. We are using the Give-to-Get protocol introduced in [9], in which a peer p discovers the forwarding rate of a child node q by periodically asking its grandchildren about the pieces received from q. In order to apply this selection mechanism, each peer needs to gather information from other nodes, which is more costly than the direct reciprocity.

- *Random* (R), in case the peers do not follow any incentive scheme, they select peers to upload to in a random fashion.

3.3 Quantification

In order to apply the DSA approach we need to define the quantification measures for Performance, Robustness and Aggressiveness. In traditional file sharing scenario this measure can simply be the downloading rate. However, in the VoD context one should use much more informative measures. We are thus using the following two measures:

- *Continuity index*, defined as the ratio of pieces received before their deadline over the total number of pieces [17], and
- *Startup delay*, defined as the time a user has to wait until the playback can be started.

As it was argued in [7] the smooth playback continuity is of higher importance in a VoD system regarding QoS. Especially, when the bandwidth is insufficient to meet a demand, it is more important to serve those peers that have already started playing. Only when there is enough bandwidth available, the startup delay should be the goal to be minimized. Habib *et al.* [8] demonstrated that when the continuity index of a stream is 95%, a users overall perceived video quality is very good.

4 Simulation Model and Methodology

Our simulation model is based on the MSR BitTorrent simulator [3], which was extended by Lucia D'Acunto with the goal of supporting VoD simulations including the piece selection policies and peer selection policies listed in Section 3. For further details regarding the extension see [7]. The original simulator includes most of the elements of a BitTorrent swarm from the overlay level down to the piece exchange of the peers. The original version of the simulator was used, and extended accordingly, in other P2P VoD studies, e.g. [19, 20].

We simulated a VoD swarm in which the service provider consists of one original seeder who is always online and has upload capacity U_s. Peers are entering the swarm based on a prescribed arrival rate $\lambda = 0.05$ peers/s, and leave the system as soon as they finish with the downloading. The media file, which is divided into n pieces of identical size, has playback rate R and size F. The average upload capacity of the participating peers is μ, and as usual in analysis of BitTorrent-like systems, we assume that the download capacity of peers is infinite. As it was laid out in [7], the required server bandwidth U_s can be calculated as $U_s = (1 - \frac{\mu}{\gamma R})\lambda F$, hence, given the parameters of the system, one only needs to set up the parameter γ that suits the needs of the participating peers. For better comparisons all the parameters used in our subsequent simulations are the same as it was in [7]. These are listed in Table 1.

Table 1: Simulation Settings

Parameter	Value
Video playback rate R	800 kb/s
Video length L	50 min
Simulation time	between 250 and 750 min ($5L$ and $15L$)
Piece size P_S	256 kB
Priority set size h	25 (direct) / 20 (indirect)
θ	2 (both direct and indirect)
Window size w	40 (direct) / 30 (indirect)
Initial buffer B	20 pieces (PROB) / w (WIN) / h (PRIO)
Upload rate μ	1000 kb/s ($1.25R$)
γ	1.3

In order to conduct the Robustness and Aggressiveness tests of DSA, further extensions of the simulator were needed. Specifically, the possibility of using two or more different strategies in a swarm was not possible in general. In our current extension the peer- and piece selection policies are private to the peers. Thus it is possible to set up experiments in which a swarm is composed by peers using different protocol variants at the same time.

Technically, the Robustness test was done in the following way. Each protocol variant is pitted against every other variant. The competition in which two different protocol variants (say p_1 and p_2) are playing against each other is called *encounter*. In each encounter peers are arriving to the swarm with a prescribed arriving rate (which was 0.05 peers/s in all the simulations). Upon arrival a peer is associated to protocol p_1 or p_2 with equal probability. For each encounter we made ten runs. For each run, we compare the average performance (measured either by Continuity index or by Startup delay) of protocol p_1 with the average performance of the other protocol. If the performance of p_1 is better than the performance of the other protocol, we mark it as a *win* for p_1, otherwise we mark it as a *loss* for p_1. The Robustness value for a protocol is calculated by the number of encounters it wins against all opponents in all encounters divided by the total number of encounters that it plays, which is a constant for all protocol variants. Concerning Aggressiveness the same kind of scheme is applied with the difference that upon arrival into the swarm, a peer is associated to protocol p_1 with probability equal to 0.1 and to protocol p_2 with probability equal to 0.9.

5 Results

Robustness. The results of the robustness test of the different VoD protocol variants are shown in Figure 1. It is important to note that the axes represent the applied measures in a normalized manner. While in the original definition of Startup delay, the lower is the better, here higher values mean more robust variant.

By its original definition the value of the Continuity index lies in the interval $[0, 1]$. This also holds here, however, the meaning of the actual value is different.

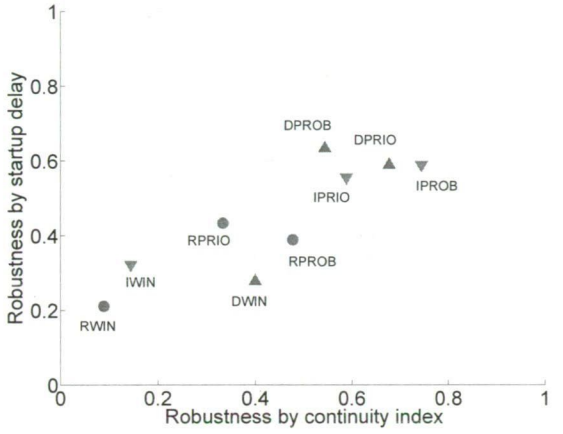


Figure 1: Robustness scatter plot

It can be clearly seen that the robustness values of the different variants are highly correlated in the two applied measures. The Pearson's linear correlation coefficient is 0.8568, In protocol-homogeneous swarms, where each peer applies the same protocol it was shown that shorter startup delay usually leads to lower continuity index [7]. What we can see here is that this does not hold in a mixed environment. The most robust variants are using PROB piece selection together with direct (for better startup) or indirect (for better continuity) reciprocity. At the lower end, random peer selection together with WIN is the worst choice. In general we can also conclude that the window-based piece selection is not robust at all. Applying random peer selection policy is also proven to be a bad choice.

In order to see more details on the actual results of the encounters using the two different measures heatmaps are shown on Figures 2 and 3. Here, lighter square means more frequent winnings against the other protocol. Regarding continuity index, it can be clearly seen that the protocol variants using Direct or Indirect reciprocity are dominating the others. This is indicated by the light upper right 5×5 corner. From here, we can see that D and I type protocols outperform all the R types and IWIN (with the exception of DWIN being outperformed by RPROB). Especially, as we have already seen, the row of IPROB is the lightest one, that is the most robust variant. This finding emphasizes the advantage of the usage of non-random incentive mechanism in P2P VoD systems. Regarding IWIN, it is interesting to remark that, according to [7] it performs well in protocol-homogeneous swarm as well as in bandwidth-heterogeneous swarm.

Using startup delay as measure the corresponding heatmap is shown on Figure 3. Due to the already noticed correlation with the other measure, we have similar picture here, although the squares are bit darker in general (according to Figure 1

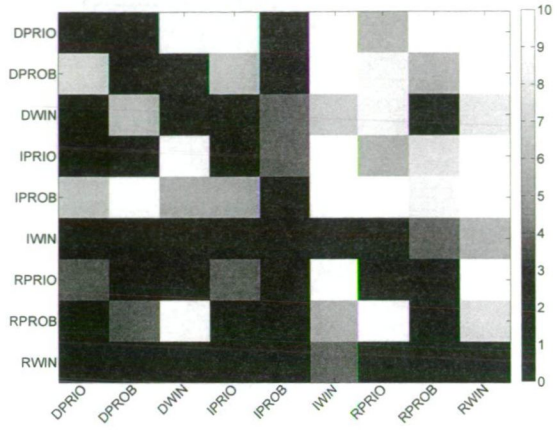


Figure 2: Heatmap of encounters with continuity index measure

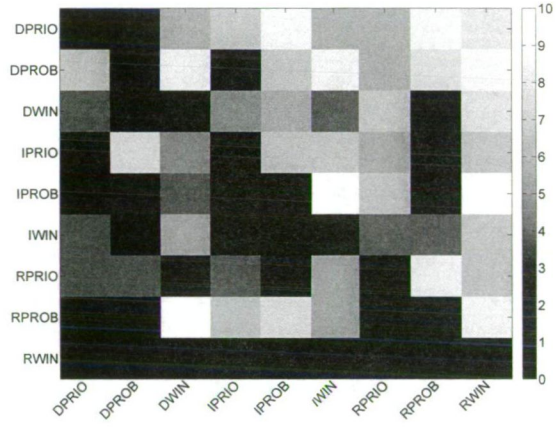


Figure 3: Heatmap of encounters with startup delay measure

the most robust protocol w.r.t. startup delay has robustness value 0.62).

Robustness versus Performance. As it was already mentioned the Performance test of the considered VoD protocol variants has already been done in [7]. On Figures 4 and 5 we can see comparisons of Robustness against Performance using the two different measures. We have to emphasize again that the values in these figures are normalized. This normalization was already explained about Robustness. As for Performance, the protocol having the highest continuity index (or

the lowest startup delay) has value 1 and all the other protocols values are relative to this.

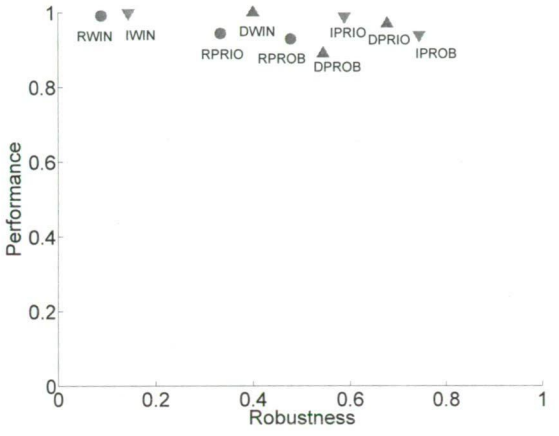


Figure 4: Robustness against Performance for continuity index

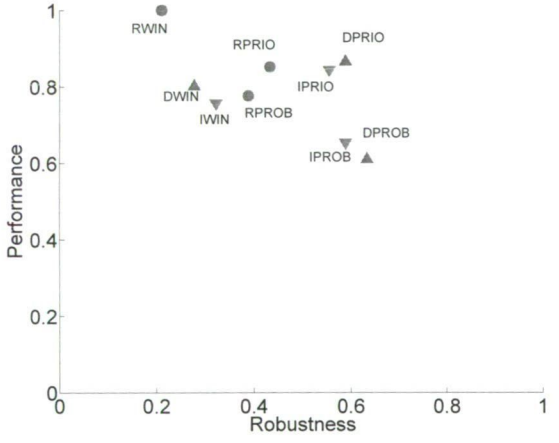


Figure 5: Robustness against Performance for startup delay

Regarding continuity index we obtained no correlations between Performance and Robustness (Figure 4) due to the simple fact that all the tested protocols have very high continuity index. This is not the case with the startup delay measure, see Figure 5. Although we did not get significant correlation, the Performance values show greater variety in this case. From these two figures we can conclude that DPRIO is the best choice to achieve the best Performance and Robustness

together. This is in line with the experiments done in [7], where the same protocols were pitted against peers using the traditional BitTorrent (e.g. file transfer instead of media streaming) and resulted in the same conclusion.

Robustness versus Aggressiveness. Now we turn our attention to the results of the Aggressiveness tests. For the continuity index the results are shown on Figure 6. The Pearson's linear correlation coefficient is 0.4553, thus, as it can be noticed also from the figure, the two measures are not correlated in general. The best choice in this case is the DPRIO protocol variant. In case of PROB piece selection there is a trade-off between Robustness (combined with indirect reciprocity) and Aggressiveness (combined with direct reciprocity). Note that protocols applying Indirect reciprocity resulted in very low Aggressiveness values. In this situation, being in the minority, the peers using indirect reciprocity are struggling with finding honest peers, about which they cannot obtain information from peers using direct reciprocity (who are in the majority). We can also see that Random peer selection is not a good choice.

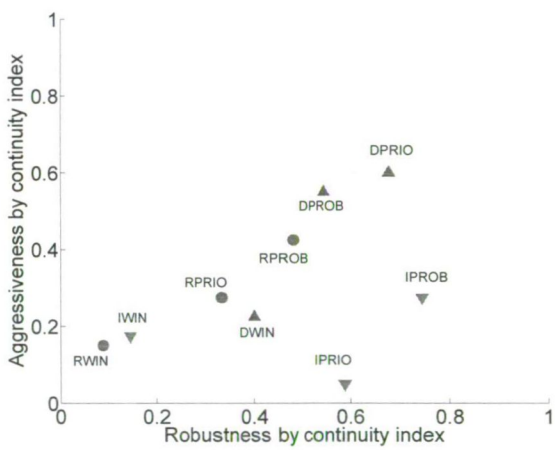


Figure 6: Robustness against Aggressiveness measured by continuity index

Regarding startup delay we get slightly different results, see Figure 7. In general, there is no correlation (correlation coefficient is 0.3376) between the two measures. In this case the DPROB variant is the best choice for maximizing the two measures.

6 Related work

The theoretical and practical (both simulation-based and measurement-based) investigation of BitTorrent-based VoD approaches have attracted the attention of many researchers in the recent years. In particular, considerable effort has been

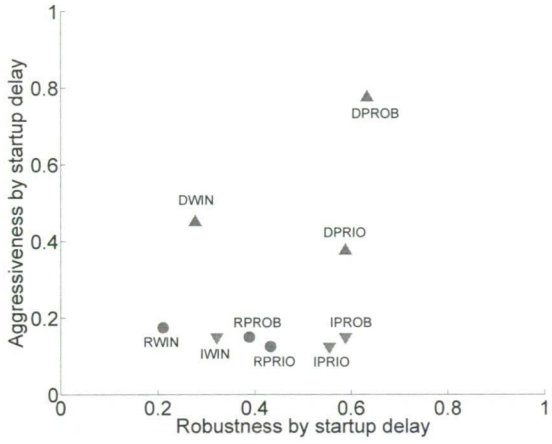


Figure 7: Robustness against Aggressiveness measured by startup delay

invested in designing and evaluating the piece selection (e.g. [1, 2, 5, 14]) and peer selection (e.g. [10, 15, 20]) policies of these systems. Perhaps the most related works to ours are the following two. Xu *et al.* [18] investigated the performance of different piece- and peer selection policies, as well as solutions to replica management. However, they did not take mixed swarms into consideration. Ma *et al.* [11] studied the performance of some piece selection policies for VoD in the presence of traditional BitTorrent peers (independently, similar simulation results can also be found in [7]). Our present work provides insights on the performance of some of the VoD approaches in mixed environment, which situation is very typical in open distributed systems.

7 Conclusion

We have demonstrated, by means of simulations, that performance of BitTorrent-like VoD approaches can show high variance depending on the protocol composition of the swarm they participate in. Following the terminologies of Design Space Analysis [16], robustness of a given protocol indicates the ability of outperforming the other protocols also existing in the same system. Our results show the importance of built-in incentive mechanisms (contrasted with random peer selection), and that probabilistic piece selection policy provides good robustness value coupled either with direct or indirect reciprocity. However, when a protocol is in the minority, direct reciprocity with priority based piece selection (for smooth continuity) or with probabilistic piece selection (for shorter startup delay) should be used.

Acknowledgements

This work was partially supported by the European Union and the European Social Fund through project FuturICT.hu (grant no.: TAMOP-4.2.2.C-11/1/KONV-2012-0013) and by the Bolyai Scholarship of the Hungarian Academy of Sciences.

References

[1] A. Vlavianos, M. Iliofotou and M. Faloutsos: BiToS: Enhancing BitTorrent for Supporting Streaming Applications. In: Proceedings of IEEE Global Internet Symposium, pp. 1–6, 2006.

[2] Annapureddy, S., Guha, S., Gkantsidis, C., Gunawardena, D., Rodriguez, P.R.: Is high-quality vod feasible using p2p swarming? In: Proceedings of the 16th international conference on World Wide Web, pp. 903–912, 2007.

[3] A.R. Bharambe, C. Herley and V.N. Padmanabhan: Analyzing and Improving a BitTorrent Network's Performance Mechanisms. In: Proceedings of IEEE INFOCOM, pp. 1–12, 2006.

[4] Axelrod, R.: The Evolution of Cooperation. Basic Books, New York, 1984.

[5] Choe, Y.R., Schuff, D.L., Dyaberi, J.M., Pai, V.S.: Improving vod server efficiency with bittorrent. In: Proceedings of the 15th International Conference on Multimedia, pp. 117–126, 2007.

[6] Cohen, B.: Incentives Build Robustness in BitTorrent. In: Workshop on Economics of Peer-to-Peer Systems. pp. 68–72, 2003.

[7] DAcunto, L., Chiluka, N., Vinkó, T., Sips, H.: Bittorrent-like P2P approaches for VoD: A comparative study. Computer Networks **57**(5), 1253 – 1276, 2013.

[8] Habib, A., Chuang, J.: Service Differentiated Peer Selection: An Incentive Mechanism for Peer-to-Peer Media streaming. IEEE Transactions on Multimedia **8**, 610–621, 2006.

[9] J.J.D. Mol, J. A. Pouwelse, M. Meulpolder, D.H.J. Epema and H.J. Sips: Give-to-Get: Free-riding-resilient Video-on-Demand in P2P Systems. In: Proceedings of SPIE MMCN, 2008.

[10] L. D'Acunto, N. Andrade, J.A. Pouwelse and H.J. Sips: Peer Selection Strategies for Improved QoS in Heterogeneous BitTorrent-like VoD Systems. In: Proceedings of the IEEE International Symposium on Multimedia, 89–96, 2010.

[11] Ma, Z., Xu, K., Liu, J., Wang, H.: Measurement, modeling and enhancement of bittorrent-based vod system. Computer Networks **56**(3), 1103 – 1117, 2012.

[12] N. Carlsson, D. L. Eager and A. Mahanti: Peer-assisted on-demand Video Streaming with Selfish Peers. In: Proceedings of IFIP NETWORKING, 586–599, 2009.

[13] N. Parvez, and C. Williamson and A. Mahanti and R. Carlsson: Analysis of BitTorrent-like Protocols for On-Demand Stored Media Streaming. In: Proceedings of ACM SIGMETRICS, 301–312, 2008.

[14] P. Savolainen, N. Raatikainen and S. Tarkoma: Windowing BitTorrent for Video-on-Demand: Not All is Lost with Tit-for-Tat. In: Proceedings of IEEE GLOBECOM, pp. 1–6, 2008.

[15] P. Shah and J. F. Pris: Peer-to-Peer Multimedia Streaming using BitTorrent. In: Proceedings of IEEE IPCCC, pp. 340 - 347, 2007.

[16] Rahman, R., Vinkó, T., Hales, D., Pouwelse, J., Sips, H.: Design space analysis for modeling incentives in distributed systems. In: Proceedings of ACM SIGCOMM, pp. 182–193, 2011.

[17] X. Zhang, J. Liu, B. Li, and T.S. P. Yum: DONet/CoolStreaming: A Data-driven Overlay Network for Live Media Streaming. In: Proceedings of IEEE INFOCOM, pp. 2102–2111, 2005.

[18] Xu, K., Liu, X., Ma, Z., Zhong, Y., Chen, W.: Exploring the policy selection of the p2p vod system: A simulation-based research. Peer-to-Peer Networking and Applications **8**(3), 459–473, 2014.

[19] Y. Borghol, S. Ardon, N. Carlsson and A. Mahanti: Toward Efficient On-Demand Streaming with BitTorrent. In: Proceedings of IFIP Networking, pp. 53–66, 2010.

[20] Yang, Y., Chow, A., Golubchik, L., Bragg, D.: Improving QoS in BitTorrent-like VoD systems. In: Proceedings of IEEE INFOCOM, pp. 1–9, 2010.

Received 29th July 2014

Acta Cybernetica **22** (2016) 687–701.

On Nonpermutational Transformation Semigroups with an Application to Syntactic Complexity*

Szabolcs Iván† and Judit Nagy-György†

Abstract

We give an upper bound of $n((n-1)!-(n-3)!)$ for the possible largest size of a subsemigroup of the full transformational semigroup over n elements consisting only of nonpermutational transformations. As an application we gain the same upper bound for the syntactic complexity of (generalized) definite languages as well.

1 Introduction

A language is generalized definite if membership can be decided for a word by looking at its prefix and suffix of a given constant length. Generalized definite languages and automata were introduced by Ginzburg [6] in 1966 and further studied in e.g. [4, 5, 13, 15]. This language class is strictly contained within the class of star-free languages, lying on the first level of the dot-depth hierarchy [1]. This class possess a characterization in terms of its syntactic semigroup [12]: a regular language is generalized definite if and only if its syntactic semigroup is locally trivial if and only if it satisfies a certain identity $x^\omega y x^\omega = x^\omega$. This characterization is hardly efficient by itself when the language is given by its minimal automaton, since the syntactic semigroup can be much larger than the automaton (a construction for a definite language with state complexity – that is, the number of states of its minimal automaton – n and syntactic complexity – that is, the size of the transition semigroup of its minimal automaton – $\lfloor e(n-1)! \rfloor$ is explicit in [2]). However, as stated in [14], Sec. 5.4, it is usually not necessary to compute the (ordered) syntactic semigroup but most of the time one can develop a more efficient algorithm by analyzing the minimal automaton. As an example for this line of research, recently, the authors of [9] gave a nice characterization of minimal automata of piecewise testable languages, yielding a quadratic-time decision algorithm, matching an alternative (but of course equivalent) earlier (also quadratic) characterization of [17] which improved the $\mathcal{O}(n^5)$ bound of [16].

*Both authors were supported by the European Union and the State of Hungary, co-financed by the European Social Fund in the framework of TÁMOP 4.2.4.A/2-11-1-2012-0001 National Excellence Program. Szabolcs Iván was also supported by NKFI grant number K108448.

†University of Szeged, Hungary, E-mail: {szabivan@inf,ngyj@math}.u-szeged.hu

DOI: 10.14232/actacyb.22.3.2016.9

There is an ongoing line of research for syntactic complexity of regular languages. In general, a regular language with state complexity n can have a syntactic complexity of n^n, already in the case when there are only three input letters. There are at least two possible modifications of the problem: one option is to consider the case when the input alphabet is binary (e.g. as done in [7, 10]). The second option is to study a strict subclass of regular languages. In this case, the syntactic complexity of a class $\mathcal{C}$ of languages is a function $n \mapsto f(n)$, with $f(n)$ being the maximal syntactic complexity a member of $\mathcal{C}$ can have whose state complexity is (at most) n. The syntactic complexity of several language classes, e.g. (co)finite, reverse definite, bifix–, factor– and subword-free languages etc. is precisely determined in [11]. However, the exact syntactic complexity of the (generalized) definite languages and that of the star-free languages (as well as the locally testable or the locally threshold testable languages) is not known yet.

In this note we give an upper bound for the maximal size of a subsemigroup of T_n, the transformation semigroup of $\{1,\ldots,n\}$, consisting of "nonpermutational" transformations only. These are exactly the (transformation) semigroups satisfying the identity $yx^\omega = x^\omega$. It is known that a language is definite iff its syntactic semigroup satisfies the same identity; thus as a corollary we get that the same bound is also an upper bound for the syntactic complexity of definite languages.

We also give a forbidden pattern characterization for the generalized definite languages in terms of the minimal automaton, and analyze the complexity of the decision problem whether a given automaton recognizes a generalized definite language, yielding an **NL**-completeness result (with respect to logspace reductions) as well as a deterministic decision procedure running in $\mathcal{O}(n^2)$ time (on a RAM machine). Analyzing the structure of their minimal automata we conclude that the syntactic complexity of generalized definite languages coincide with that of definite languages.

2 Notation

When $n \geq 0$ is an integer, $[n]$ stands for the set $\{1,\ldots,n\}$. For the sets A and B, A^B denotes the set of all functions $f : B \to A$. When $f \in A^B$ and $C \subseteq B$, then $f|_C \in A^C$ denotes the restriction of f to C. When $A_1,\ldots,A_n$ are disjoint sets, A is a set and for each $i \in [n]$, $f_i : A_i \to A$ is a function, then the *source tupling* of $f_1,\ldots,f_n$ is the function $[f_1,\ldots,f_n] : \left(\bigcup_{i\in[n]} A_i\right) \to A$ with $a[f_1,\ldots,f_n] = af_i$ for the unique i with $a \in A_i$.

T_n is the transformation semigroup of $[n]$ (i.e. $[n]^{[n]}$), where composition is understood as $p(fg) := (pf)g$ for $p \in [n]$ and $f, g : [n] \to [n]$ (i.e., transformations of $[n]$ act on $[n]$ from the right to ease notation in the automata-related part of the paper). Elements of T_n are often written as n-ary vectors as usual, e.g. $f = (1,3,3,2)$ is the member of T_4 with $1f = 1$, $2f = 3$, $3f = 3$ and $4f = 2$.

When $f : A \to A$ is a transformation of a set A, and X is a subset of A, then Xf denotes the subset $\{xf : x \in X\}$ of A.

A transformation $f : A \to A$ of a (finite) set A is *nonpermutational* if $Xf = X$ implies $|X| = 1$ for any nonempty $X \subseteq A$. Otherwise it's *permutational*. NP_n stands for the set of all nonpermutational transformations of $[n]$.

Another class of functions used in the paper is that of the *elevating* functions: for the integers $0 < k \leq n$, a function $f : [k] \to [n]$ is elevating if $i \leq if$ for each $i \in [k]$ with equality allowed only in the case when $i = n$ (note that this also implies $k = n$ as well).

We assume the reader is familiar with the standard notions of automata and language theory, but still we give a summary for the notation.

An *alphabet* is a nonempty finite set Σ. The set of *words* over Σ is denoted Σ^*, while Σ^+ stands for the set of *nonempty words*. The *empty word* is denoted ε. A *language* over Σ is an arbitrary set $L \subseteq \Sigma^*$ of Σ-words.

A (finite) *automaton* (over Σ) is a system $\mathbb{A} = (Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of states, $q_0 \in Q$ is the start state, $F \subseteq Q$ is the set of final (or accepting) states, and $\delta : Q \times \Sigma \to Q$ is the transition function. The transition function δ extends in a unique way to a right action of the monoid Σ^* on Q, also denoted δ for ease of notation. When δ is understood, we write $q \cdot u$, or simply qu for $\delta(q, u)$. Moreover, when $C \subseteq Q$ is a subset of states and $u \in \Sigma^*$ is a word, let Cu stand for the set $\{pu : p \in C\}$ and when L is a language, $CL = \{pu : p \in C, u \in L\}$. The *language recognized by* $\mathbb{A}$ is $L(\mathbb{A}) = \{x \in \Sigma^* : q_0 x \in F\}$. A language is *regular* if it can be recognized by some finite automaton.

The state $q \in Q$ is *reachable* from a state $p \in Q$ in $\mathbb{A}$, denoted $p \preceq_{\mathbb{A}} q$, or just $p \preceq q$ if there is no danger of confusion, if $pu = q$ for some $u \in \Sigma^*$. An automaton is *connected* if its states are all reachable from its start state.

Two states p and q of $\mathbb{A}$ are *distinguishable* if there exists a word $u \in \Sigma^*$ such that exactly one of pu and qu belongs to F. In this case we say that u *separates* p and q. A connected automaton is called *reduced* if each pair of distinct states is distinguishable.

It is known that for each regular language L there exists a reduced automaton, unique up to isomorphism, recognizing L. This automaton $\mathbb{A}_L$ can be computed from any automaton recognizing L by an efficient algorithm called minimization and is called the *minimal automaton* of L.

The classes of the equivalence relation $p \sim q \Leftrightarrow p \preceq q$ and $q \preceq p$ are called *components* of $\mathbb{A}$. A component C is *trivial* if $C = \{p\}$ for some state p such that $pa \neq p$ for any $a \in \Sigma$, and is a *sink* if $C\Sigma \subseteq C$. It is clear that each automaton has at least one sink and sinks are never trivial. The *component graph* $\Gamma(\mathbb{A})$ of $\mathbb{A}$ is an edge-labelled directed graph (V, E, ℓ) along with a mapping $c : Q \to V$ where V is the set of the $\sim$-classes of $\mathbb{A}$, the mapping c associates to each state q its class $q/\sim = \{p : p \sim q\}$ and for two classes $p/\sim$ and $q/\sim$ there exists an edge from $p/\sim$ to $q/\sim$ labelled by $a \in \Sigma$ if and only if $p'a = q'$ for some $p' \sim p$, $q' \sim q$. It is known that the component graph can be constructed from $\mathbb{A}$ in linear time. Note that the mapping c is redundant but it gives a possibility for determining whether $p \sim q$ holds in constant time on a RAM machine, provided $Q = [n]$ for some $n > 0$ and c is stored as an array.

When $\mathbb{A} = (Q, \Sigma, \delta, q_0, F)$ is an automaton, its *transformation semigroup* $\mathcal{T}(\mathbb{A})$

consists of the set of transformations of Q induced by nonempty words, i.e. $\mathcal{T}(\mathbb{A}) = \{u^{\mathbb{A}} : u \in \Sigma^+\}$ where $u^{\mathbb{A}} : Q \to Q$ is the transformation defined as $q \mapsto qu$. The *state complexity* $\mathrm{stc}(L)$ of a regular language L is the number of states of its minimal automaton $\mathbb{A}_L$ while its *syntactic complexity* $\mathrm{syc}(L)$ is the cardinality of its transformation semigroup $\mathcal{T}(\mathbb{A}_L)$. The *syntactic complexity* of a *class* of languages C is a function $f : \mathbb{N} \to \mathbb{N}$ defined as

$$f(n) = \max\{\mathrm{syc}(L) : L \in C, \mathrm{stc}(L) \leq n\},$$

i.e. $f(n)$ is the maximal size that the transformation semigroup of a minimal automaton of a language belonging to C can have, provided the automaton has at most n states.

3 Semigroups of nonpermutational transformations

Observe that NP_n is not a semigroup (i.e., not closed under composition) when $n > 2$. Indeed, if $f = (2, 3, 3)$ and $g = (1, 1, 2)$ (both being nonpermutational), then their product $fg = (1, 2, 2)$ is permutational with $\{1, 2\}fg = \{1, 2\}$. (See Figure 1.)

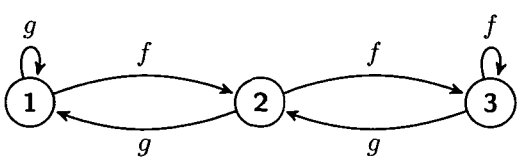


Figure 1: f and g are nonpermutational, fg is permutational

Thus, the following question is nontrivial: how large a subsemigroup of T_n, which consists only of nonpermutational transformations can be? The obvious upper bound is n^n, the size of T_n.

As a first step we give an upper bound of n^{n-2}. Observe that the following are equivalent for a function $f : [n] \to [n]$:

i) f is nonpermutational;

ii) the graph of f is a rooted tree with edges directed towards the root, and with a loop edge attached on the root;

iii) f^ω, the unique idempotent power of f is a constant function.

Here "the graph of f" is of course the directed graph Γ_f on vertex set $[n]$ and with (i, j) being an edge iff $if = j$.

Indeed, assume f is nonpermutational. Let X be the set of all nodes of Γ_f lying on some closed path. (Since each node of the finite graph Γ_f has outdegree 1, X is nonempty.) Then $Xf = X$, thus $|X| = 1$, i.e. f has a unique fixed point $\mathrm{Fix}(f)$

and apart from the loop edge on $\mathrm{Fix}(f)$, Γ_f is a directed acyclic graph (DAG) with each node distinct from $\mathrm{Fix}(f)$ having outdegree 1 – that is, a tree rooted at $\mathrm{Fix}(f)$, with edges directed towards the root, showing i) $\to$ ii). Then f^n is a constant function with value $\mathrm{Fix}(f)$, showing ii) $\to$ iii); finally, if $Xf = X$ for some nonempty $X \subseteq [n]$, then $Xf^\omega = X$, showing $|X| = 1$ since the image of f^ω is a singleton.

Now from ii) we get that the members of NP_n are exactly the rooted trees with edges directed towards the root on which a loop edge is attached – we call such a graph an inverted looped arborescence, or ILA for short. By Cayley's theorem on the number of labeled rooted trees over n nodes, the number of all ILAs (i.e., $|NP_n|$) is n^{n-2}, giving a slightly better upper bound.

To achieve an upper bound of $n!$, suppose $S \subseteq NP_n$ is a subsemigroup of T_n. For $i \in [n]$, let $S_i \subseteq S$ be the subsemigroup $\{f \in S : \mathrm{Fix}(f) = i\}$ of S. Note that S_i is indeed a semigroup: by assumption, S is closed under composition and consists of nonpermutational transformations only, moreover, if i is the common (unique) fixed point of f and g, then it is also a fixed point of fg as well, thus S_i is closed under composition.

We give an upper bound of $(n-1)!$ **for** $|S_i|$, $i \in [n]$, yielding $|S| \leq n!$. To this end, let Γ_i be the graph on vertex set $[n]$ with (j,k) being an edge iff $jf = k$ for some $f \in S_i$. Then, apart from the trivial case when $S_i = \emptyset$, (i,i) is an edge in Γ_i, moreover i is a sink (since $if = i$ for each $f \in S_i$). Note that in the case when $S_i = \emptyset$, $|S_i| = 0 \leq (n-1)!$ clearly holds. Observe that Γ_i is transitive, since if (j,k) and (k,ℓ) are edges of Γ_i, then $jf = k$ and $kg = \ell$ for some $f, g \in S_i$; since S_i is a semigroup, fg is also in S_i thus (j,ℓ) is also an edge in Γ_i. Now assume some node $j \in [n]$ is in a nontrivial strongly connected component (SCC) of Γ_i, i.e. j lies on some closed path. By transitivity, (j,j) is an edge of Γ_i, thus $jf = j$ for some $f \in S_i$, thus $j = i$ since $i = \mathrm{Fix}(f)$ is the unique fixed point of $f \in S_i$. Hence by dropping the edge (i,i) we get a DAG again, thus Γ_i (viewed as a relation) is a strict partial ordering of $[n]$ with largest element i. Let $\prec_i$ stand for this partial ordering, i.e., let $j \prec_i k$ if and only if $j \neq i$ and $jf = k$ for some $f \in S_i$. Let us also fix some arbitrary total ordering $<_i$ extending $\prec_i$ and write the members of $[n]$ in the order $a_{i,1} <_i a_{i,2} <_i \ldots <_i a_{i,n} = i$. Then for any $f \in S_i$ and $1 \leq j < n$ we have $a_{i,j} <_i a_{i,j}f$, and $a_{i,n}f = a_{i,n}$. Since the number of functions $f : [n] \to [n]$ satisfying this constraint is $(n-1)!$ ($a_{i,1}$ can get $(n-1)$ different possible values, $a_{i,2}$ can get $(n-2)$ etc.), we immediately get $|S_i| \leq (n-1)!$ as well, yielding $|S| \leq n!$.

Via a somewhat cumbersome case analysis we can sharpen this upper bound to $n((n-1)! - (n-3)!)$. Without loss of generality assume that S_n is (one of) the largest of the semigroups S_i and that $<_n$ is the usual ordering $<$ of $[n]$ (we can achieve this by a suitable bijection).

Lemma 1. *Suppose for each $i < j$ and $k < \ell$ with $i \neq k$ there exists a function $f \in S_n$ with $if = j$ and $kf = \ell$.*

Then the following holds for each $i, j \in [n]$ and $f \in S_i$:

i) if $j < i$, then $j < jf$;

ii) if $i \leq j$, then $jf = i$.

Proof. By assumption, the statements clearly hold for $i = n$. Let $i < n$ be arbitrary and $f \in S_i$ a transformation. Clearly $if = i$ by the definition of S_i. Also, $nf < n$ since $i \neq n$ is the unique fixed point of f.

Suppose $jf < j$ for some j. Then $jf = nf$ has to hold: if $jf \neq nf$, then by assumption $jfg = j$ and $nfg = n$ for some $g \in S_n$, thus both j and n are distinct fixed points of fg, a contradiction. (See Figure 2.) This implies in particular that $j \leq jf$ for each $j < nf$.

Also, if $nf < i$, then $nfg = i$ and $ig = n$ for some $g \in S_n$, in which case $fgfg$ has two distinct fixed points n and i, a contradiction. (See Figure 2.) Thus $i \leq nf$.

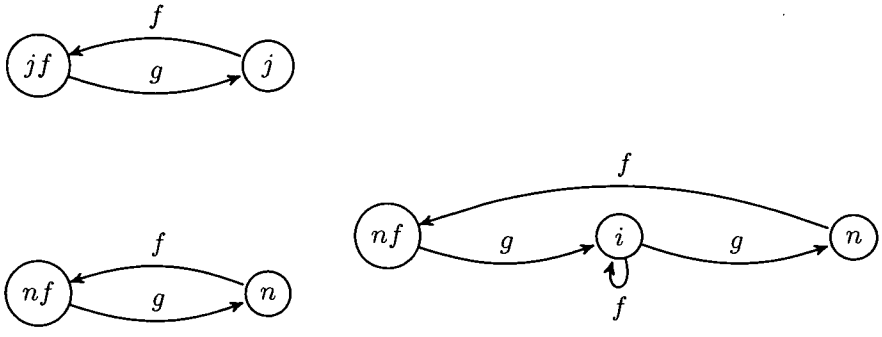


Figure 2: Left: if $jf < j$, $jf \neq nf$, then fg has two fixed points. Right: If $nf < i$, then $fgfg$ has two fixed points

Assume $i < nf$. Then (since $nf^n = i < nf$) there is some $k > 0$ such that $nf^{k+1} < nf$. If k is chosen to be the smallest possible such k, then $nf \leq nf^k$, yielding $(nf^k)f < nf \leq nf^k$, a contradiction (by $(nf^k)f < nf^k$, it should hold that $(nf^k)f = nf$, see Figure 3). Hence $i = nf$ is the unique fixed point of f and for each $j < i$, $j < jf$ indeed has to hold, showing i).

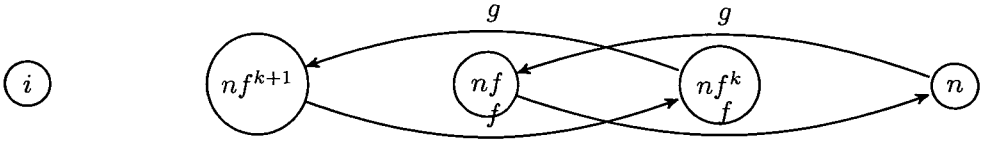


Figure 3: If $i < nf$, then fg has two distinct fixed points

Finally, assume $i < j < jf$. Then $ig = j$ and $jfg = n$ for some $g \in S_n$ (if $jf = n$, then this latter case always gets satisfied, otherwise it's by assumption on S_n), and $fgfg$ has two distinct fixed points j and n. Thus we have indeed shown that $nf = i$ is the unique fixed point of f, $j < jf$ for each $i < j$ and $jf = i$ for each $i \leq j \leq n$. □

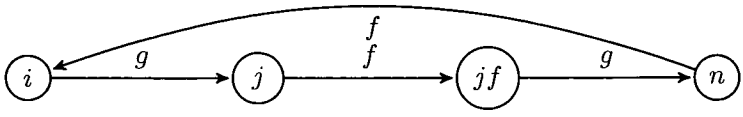


Figure 4: If $i < j < jf$, then $fgfg$ has two distinct fixed points

Lemma 1 has the following corollary:

Theorem 1. *The cardinality of any subsemigroup S of T_n consisting only of nonpermutational transformations is at most $n((n-1)! - (n-3)!)$.*

Proof. As before, let S_i stand for $\{f \in S : \mathrm{Fix}(f) = i\}$ and without loss of generality we assume that amongst them S_n is one of the largest ones, moreover $<_n$ coincides with $<$.

If for each $i < j$ and $i' < j'$ with $i \neq i'$ there is some $f \in S_n$ with $if = j$ and $i'f = j'$, then by Lemma 1 S_i can consist of at most $(n-1)(n-2)\ldots(n-i-1) = \frac{(n-1)!}{(n-i)!}$ elements (we have to choose for each $j < i$ a larger integer and that's all since the other elements have to be mapped to i). Also $|S_n| \leq (n-1)!$ as well. Summing up we get an upper bound for these semigroups

$$\sum_{i=1}^{n} \frac{(n-1)!}{(n-i)!} = (n-1)! \sum_{j=0}^{n-1} \frac{1}{j!} = \lfloor e(n-1)! \rfloor,$$

which comes from the facts that $e = \sum_{j=0}^{\infty} \frac{1}{j!}$ and $(n-1)! \sum_{j=n}^{\infty} \frac{1}{j!} < 1$.

For the other case, suppose there exist an $i < j$ and an $i' < j'$ with $i \neq i'$ such that $if = j$ and $i'f = j'$ do not both hold for any $f \in S_n$. Still, $i < if$ for each $i < n$ and $nf = n$, by definition of S_n and the assumption $<=<_n$. The number of such functions satisfying both $if = j$ and $i'f = j'$ is $\frac{(n-1)!}{(n-i)(n-j)} \geq (n-3)!$, hence the size of S_n is upper-bounded by $(n-1)! - (n-3)!$. Since S_n is the largest amongst the S_i's and S is the disjoint union of them we get the claimed upper bound $n((n-1)! - (n-3)!)$. □

We note that the construction for the first case, yielding the upper bound $\lfloor e(n-1)! \rfloor$ indeed constructs a semigroup B which is exactly the semigroup from [2] conjectured there to be a candidate for the maximal-size such subsemigroup.

Our proof can be viewed as a support for this conjecture and can be reformalized as follows: if there exists some i such that many transformations share this fixed point i, then the size of S is upper-bounded by $\lfloor e(n-1)! \rfloor$ and S is isomorphic to a subsemigroup of B. The question is, whether one can construct a larger semigroup by putting not too many functions sharing a common fixed point. We also conjecture that B is a good candidate for a maximal-size subsemigroup of T_n consisting of nonpermutational transformations only.

4 Definite and generalized definite languages

A language L is *definite* if there exists a constant $k \geq 0$ such that for any $x \in \Sigma^*$, $y \in \Sigma^k$ we have $xy \in L \Leftrightarrow y \in L$ and is *generalized definite* if there exists a constant $k \geq 0$ such that for any $x_1, x_2 \in \Sigma^k$ and $y \in \Sigma^*$ we have $x_1 y x_2 \in L \Leftrightarrow x_1 x_2 \in L$.

These are both subclasses of the star-free languages, i.e. can be built from the singletons with repeated use of the concatenation, finite union and complementation operations. It is known that the following decision problem is complete for **PSPACE**: given a regular language L with its minimal automaton, is L star-free? In contrast, the question for these subclasses above are tractable.

Minimal automata of these languages possess a characterization in terms of *forbidden patterns*. In our setting, a pattern is an edge-labelled, directed graph $P = (V, E, \ell)$, where V is the set of vertices, $E \subseteq V^2$ is the set of edges, and $\ell : E \to \mathcal{X}$ is a labelling function which assigns to each edge a variable. An automaton $\mathbb{A} = (Q, \Sigma, \delta, q_0, F)$ *admits a pattern* $P = (V, E, \ell)$ if there exists an *injective* mapping $f : V \to Q$ and a map $h : \mathcal{X} \to \Sigma^+$ such that for each $(u, v) \in E$ labelled x we have $f(u) \cdot h(x) = f(v)$. Otherwise $\mathbb{A}$ *avoids* P.

As an example, consider the pattern P_d on Figure 5.

(a) Pattern P_d. (b) Pattern P_g.

Figure 5: Patterns for definite and generalized definite languages.

4.1 Syntactic complexity of definite languages

A reduced automaton avoids P_d if and only if it recognizes a definite language. Indeed, a language L is definite iff its syntactic semigroup satisfies the identity $yx^\omega = x^\omega$. Now assume $L(\mathbb{A})$ admits P_d with $px = p$ and $qx = q$ with $p \neq q$ and $x \in \Sigma^+$. If $q_0 x^\omega = p$, then $q_0 x^\omega \neq q_0 y x^\omega$ for a (nonempty) word y with $q_0 y = q$. If $q_0 x^\omega \neq p$, then $q_0 x^\omega \neq q_0 y x^\omega$ for a (nonempty) y with $q_0 y = p$, thus the identity is not satisfied. For the other directon, if the transition semigroup of an automaton $\mathbb{A}$ does not satisfy $x^\omega = yx^\omega$, then $p_0 x_0^\omega \neq p_0 y x_0^\omega$ for some p_0, x_0 and y; choosing $p = p_0 x^\omega$, $q = p_0 y$ and $x_0 = x^\omega$ witnesses admittance of P_d. (For a more detailed discussion see e.g. [2].)

Observe that avoiding P_d is equivalent to state that each nonempty word induces a transformation with at most one fixed point, which is further equivalent to state that each nonempty word induces a non-permutational transformation: for each nonempty u, the word $u^{|Q|!}$ fixes each state belonging to a nontrivial component of the graph of u, hence u also can have only one state in a nontrivial component,

i.e. u induces a nonpermutational transformation. (Again, see [2] for a different formulation.[1].)

Thus Theorem 1 has the following byproduct:

Corollary 1. *The syntactic complexity of the definite languages is at most* $n((n-1)! - (n-3)!)$.

4.2 Syntactic complexity of generalized definite languages

In this subsection we show that the syntactic complexity of definite and generalized definite languages coincide. To this end we study the structure of the minimal automata of the members of the latter class. In the process we give a (to our knowledge) new (but not too surprising) characterization of the minimal automata of generalized definite languages, leading to an **NL**-completeness result of the corresponding decision problem, as well as a low-degree polynomial deterministic algorithm.

Our first observation is the following characterization:

Theorem 2. *The following are equivalent for a reduced automaton* $\mathbb{A}$*:*

i) $\mathbb{A}$ *avoids* P_g.

ii) Each nontrivial component of $\mathbb{A}$ *is a sink, and for each nonempty word* u *and sink* C *of* $\mathbb{A}$*, the transformation* $u|_C : C \to C$ *is non-permutational.*

iii) $\mathbb{A}$ *recognizes a generalized definite language.*

Proof. Let $\mathbb{A} = (Q, \Sigma, \delta, q_0, F)$ be a reduced automaton.

i)→ii). Suppose $\mathbb{A}$ avoids P_g. Suppose that $u|_C$ is permutational for some sink C and word $u \in \Sigma^+$. Then there exists a set $D \subseteq C$ with $|D| > 1$ such that u induces a permutation on D. Then, $x = u^{|D|!}$ is the identity on D. Choosing arbitrary distinct states $p, q \in D$ and a word y with $py = q$ (such y exists since p and q are in the same component of $\mathbb{A}$), we get that $\mathbb{A}$ admits P_g by the (p, q, x, y) defined above, a contradiction. Hence, $u|_C$ is non-permutational for each sink C and word $u \in \Sigma^+$.

Now assume there exists a nontrivial component C which is not a sink. Then, $pu = p$ for some $p \in C$ and word $u \in \Sigma^+$. Since C is not a sink, there exists a sink $C' \neq C$ reachable from p (i.e. all of its members are reachable from p). Since u induces a non-permutational transformation on C', $x = u^{|C'|}$ induces a constant function on C'. Let q be the unique state in the image of $x|_{C'}$. Since C' is reachable from p, there exists some nonempty word y such that $py = q$. Hence, $px = p$, $qx = q$, $py = q$ and $\mathbb{A}$ admits P_g, a contradiction.

ii)→iii). Suppose the condition of ii) holds. We show that $L = L(\mathbb{A})$ is generalized definite. By the assumption, q_0u belongs to a sink for any u with $|u| \geq |Q|$. On

[1]Since – up to our knowledge – [2] has not been published yet in a peer-reviewed journal or conference proceedings, we include a proof of this fact. Nevertheless, we do not claim this result to be ours, by any means.

the other hand, viewing a sink C as a (reduced) automaton $\mathbb{C} = (C, \Sigma, \delta|_C, p, F \cap C)$ with p being an arbitrary state of C we get that the transition semigroup of $\mathbb{C}$ consists of nonpermutational transformations only, i.e. $L(\mathbb{C})$ is k-definite for some $k = k_C$. Hence choosing n to be the maximum of $|Q|$ and the values k_C with C being a sink we get that L is n-generalized definite (since the length-n prefix of u determines the sink C to which q_0u belongs and the length-n suffix of u, once we know C, determines the unique state in Cu).

iii)→i). Suppose $L(\mathbb{A})$ is generalized definite. Then its syntactic semigroup satisfies $x^\omega y x^\omega = x^\omega$ (see e.g. [14]).

Now assume $\mathbb{A}_L$ admits P_g with $px = p$, $qx = q$ and $py = q$ for the nonempty words x, y and different states p, q. Then $px^\omega = p$ and $px^\omega yx^\omega = q$, and the identity is not satisfied, thus L is not generalized definite. □

In [2] it has been shown that the class of definite languages has syntactic complexity $\geq \lfloor e \cdot (n-1)! \rfloor$, thus the same lower bound also applies for the larger class of generalized definite languages.

Theorem 3. *The syntactic complexity of the definite and that of the generalized definite languages coincide.*

Proof. It suffices to construct for an arbitrary reduced automaton $\mathbb{A} = (Q, \Sigma, \delta, q_0, F)$ recognizing a generalized definite language a reduced automaton $\mathbb{B} = (Q, \Delta, \delta', q_0, F')$ for some Δ recognizing a definite language such that $|\mathcal{T}(\mathbb{A})| \leq |\mathcal{T}(\mathbb{B})|$.

By Theorem 2, if $L(\mathbb{A})$ is generalized definite and $\mathbb{A}$ is reduced, then Q can be partitioned as a disjoint union $Q = Q_0 \uplus Q_1 \uplus \ldots \uplus Q_c$ for some $c > 0$ such that each Q_i with $i \in [c]$ is a sink of $\mathbb{A}$ and Q_0 is the (possibly empty) set of those states that belong to a trivial component. Without loss of generality we can assume that $Q = [n]$ and $Q_0 = [k]$ for some n and k, and that for each $i \in [k]$ and $a \in \Sigma$, $i < ia$. The latter condition is due to the fact that reachability restricted to the set Q_0 of states in trivial components is a partial ordering of Q_0 which can be extended to a linear ordering. Clearly, if Q_0 is nonempty, then by connectedness $q_0 = 1$ has to hold; otherwise $c = 1$ and we again may assume $q_0 = 1$. Also, $Q_i\Sigma \subseteq Q_i$ for each $i \in [c]$, and let $|Q_1| \leq |Q_2| \leq \ldots \leq |Q_c|$.

Then, each transformation $f : Q \to Q$ can be uniquely written as the source tupling $[f_0, \ldots, f_c]$ of some functions $f_i : Q_i \to Q$ with $f_i : Q_i \to Q_i$ for $0 < i \leq c$. For any $[f_0, \ldots, f_c] \in \mathcal{T} = \mathcal{T}(\mathbb{A})$ the following hold: $f_0(i) > i$ for each $i \in [k]$, and f_j is non-permutational on Q_j for each $j \in [c]$. For $k = 0, \ldots, c$, let $\mathcal{T}_k$ stand for the set $\{f_k : f \in \mathcal{T}\}$ (i.e. the set of functions $f|_{Q_k}$ with $f \in \mathcal{T}$). Then, $|\mathcal{T}| \leq \prod_{0 \leq k \leq c} |\mathcal{T}_k|$.

If $|Q_c| = 1$, then all the sinks of $\mathbb{A}$ are singleton sets. Thus there are at most two sinks, since if C and D are singleton sinks whose members do not differ in their finality, then their members are not distinguishable, thus $C = D$ since $\mathbb{A}$ is reduced. Such automata recognize reverse definite languages, having a syntactic semigroup of size at most $(n-1)!$ by [2], thus in that case $\mathbb{B}$ can be chosen to an arbitrary definite automaton having n state and a syntactic semigroup of size at least $\lfloor e(n-1)! \rfloor$ (by the construction in [2], such an automaton exists). Thus we

may assume that $|Q_c| > 1$. (Note that in that case Q_c contains at least one final and at least one non-final state.)

Let us define the sets $\mathcal{T}'_k$ of functions $Q_i \to Q$ as $\mathcal{T}'_0$ is the set of all elevating functions from $[k]$ to $[n]$, $\mathcal{T}'_c = \mathcal{T}_c$ and for each $0 < k < c$, $\mathcal{T}'_k = Q_c^{Q_k}$. Since $\mathcal{T}_k \subseteq Q_k^{Q_k}$ and $|Q_k| \leq |Q_c|$ for each $k \in [c]$, we have $|\mathcal{T}_k| \leq |\mathcal{T}'_k|$ for each $0 \leq k \leq c$. Thus defining $\mathcal{T}' = \{[f_0, \ldots, f_c] : f_i \in \mathcal{T}'_i\}$ it holds that $|\mathcal{T}| \leq |\mathcal{T}'|$.

We define $\mathbb{B}$ as $(Q, \mathcal{T}', \delta', q_0, F)$ with $\delta'(q, f) = f(q)$ for each $f \in \mathcal{T}'$. We show that $\mathbb{B}$ is a reduced automaton avoiding P_d, concluding the proof.

First, observe that $\mathbb{B}$ has exactly one sink, Q_c, and all the other states belong to trivial components (since by each transition, each member of Q_0 gets elevated, and each member of Q_i with $0 < i < c$ is taken into Q_c). Hence if $\mathbb{B}$ admits P_d, then $pt = p$ and $qt = q$ for some distinct pair $p, q \in Q_c$ of states and $t = [t'_0, \ldots, t'_c] \in \mathcal{T}'$. This is further equivalent to $pt'_c = p$ and $qt'_c = q$ for some $p \neq q$ in Q_c and $t'_c \in \mathcal{T}'_c$. By definition of $\mathcal{T}'_c = \mathcal{T}_c$, there exists a transformation of the form $t = [t_0, \ldots, t_{c-1}, t'_c] \in \mathcal{T}$ induced by some word x, thus $px = p$ and $qx = q$ both hold in $\mathbb{A}$, and since p, q are in the same sink, there also exists a word y with $py = q$. Hence $\mathbb{A}$ admits P_g, a contradiction.

Second, $\mathbb{B}$ is connected. To see this, observe that each state $p \neq 1$ is reachable from 1 by any transformation of the form $t = [f_p, t_1, \ldots, t_c]$ where $f_p : [k] \to [n]$ is the elevating function with $1f_p = p$ and $if_p = n$ for each $i > 1$. Of course 1 is also trivially reachable from itself, thus $\mathbb{B}$ is connected.

Also, whenever $p \neq q$ are different states of $\mathbb{B}$, then they are distinguishable by some word. To see this, we first show this for $p, q \in Q_c$. Indeed, since $\mathbb{A}$ is reduced, some transformation $t = [t_0, \ldots, t_c] \in \mathcal{T}$ separates p and q (exactly one of $pt = pt_c$ and $qt = qt_c$ belong to F). Since $\mathcal{T}_c = \mathcal{T}'_c$, we get that p and q are also distinguishable by in $\mathbb{B}$ by any transformation of the form $t' = [t'_0, \ldots, t'_{c-1}, t_c] \in \mathcal{T}'$. Now suppose neither p nor q belong to Q_c. Then, since $\{[t'_0, \ldots, t'_{c-1}] : t'_i \in \mathcal{T}'_i\} = Q_c^{Q \setminus Q_c}$, and $|Q_c| > 1$, there exists some $t = [t'_0, \ldots, t'_{c-1}]$ with $pt \neq qt$, thus any transformation of the form $[t'_0, \ldots, t'_{c-1}, t_c] \in \mathcal{T}'$ maps p and q to distinct elements of Q_c, which are already known to be distinguishable, thus so are p and q. Finally, if $p \in Q_c$ and $q \notin Q_c$, then let $t_c \in \mathcal{T}_c$ be arbitrary and $t' = [t'_0, \ldots, t_{c-1}] \in Q_c^{Q \setminus Q_c}$ with $qt' \neq pt_c$. Then $[t', t_c]$ again maps p and q to distinct states of Q_c.

Thus $\mathbb{B}$ is reduced, concluding the proof: $\mathbb{B}$ is a reduced automaton recognizing a definite language and having a syntactic semigroup $\mathcal{T}'$ with $|\mathcal{T}'| \geq |\mathcal{T}|$. □

4.3 Complexity issues

Using the characterization given in Theorem 2, we study the complexity of the following decision problem GENDEF: given a finite automaton $\mathbb{A}$, is $L(\mathbb{A})$ a generalized definite language?

Theorem 4. *Problem* GENDEF *is* **NL***-complete.*

Proof. First we show that GENDEF belongs to **NL**. By [3], minimizing a DFA can be done in nondeterministic logspace. Thus we can assume that the input is already

minimized, since the class of (nondeterministic) logspace computable functions is closed under composition.

Consider the following algorithm:

1. Guess two different states p and q.
2. Let $s := p$.
3. Guess a letter $a \in \Sigma$. Let $s := sa$.
4. If $s = q$, proceed to Step 5. Otherwise go back to Step 3.
5. Let $p' := p$ and $q' := q$.
6. Guess a letter $a \in \Sigma$. Let $p' := p'a$ and $q' = q'a$.
7. If $p = p'$ and $q = q'$, accept the input. Otherwise go back to Step 6.

The above algorithm checks whether $\mathbb{A}$ admits P_g: first it guesses $p \neq q$, then in Steps 2–4 it checks whether q is accessible from p, and if so, then in Steps 5–7 it checks whether there exists a word $x \in \Sigma^+$ with $px = p$ and $qx = q$. Thus it decides[2] the complement of GENDEF, in nondeterministic logspace; since $\mathbf{NL} = \text{co}\mathbf{NL}$, we get that GENDEF $\in \mathbf{NL}$ as well.

For **NL**-completeness we recall from [8] that the reachability problem for DAGs (DAG-REACH) is complete for **NL**: given a directed acyclic graph $G = (V, E)$ on $V = [n]$ with $(i, j) \in E$ only if $i < j$, is n accessible from 1? We give a logspace reduction from DAG-REACH to GENDEF as follows. Let $G = ([n], E)$ be an instance of DAG-REACH. For a vertex $i \in [n]$, let $N(i) = \{j : (i, j) \in E\}$ stand for the set of its neighbours and let $d(i) = |N(i)| < n$ denote the outdegree of i. When $j \in [d(i)]$, then the jth neighbour of i, denoted $n(i, j)$ is simply the jth element of $N(i)$ (with respect to the usual ordering of integers of course). Note that for any $i \in [n]$ and $j \in [d(i)]$ both $d(i)$ and the $n(i, j)$ (if exists) can be computed in logspace.

We define the automaton $\mathbb{A} = ([n+1], [n], \delta, 1, \{n+1\})$ where

$$\delta(i,j) = \begin{cases} n+1 & \text{if } (i = n+1) \text{ or } (j = n) \text{ or } (i < n \text{ and } d(i) < j); \\ 1 & \text{if } i = n \text{ and } j < n; \\ n(i,j) & \text{otherwise.} \end{cases}$$

Note that $\mathbb{A}$ is indeed an automaton, i.e. $\delta(i, j)$ is well-defined for each i, j.

We claim that $\mathbb{A}$ admits P_g if and only if n is reachable from 1 in G. Observe that the underlying graph of $\mathbb{A}$ is G, with a new edge $(n, 1)$ and with a new vertex $n + 1$, which is a neighbour of each vertex. Hence, $\{n + 1\}$ is a sink of $\mathbb{A}$ which is reachable from all other states. Thus $\mathbb{A}$ admits P_g if and only if there exists

[2]Note that in this form, the algorithm can enter an infinite loop which fits into the definition of nondeterministic log*space*. Introducing a counter and allowing at most n steps in the first cycle and at most n^2 in the second we get a nondeterministic algorithm using logspace and polytime, as usual.

a nontrivial component of $\mathbb{A}$ which is different from $\{n+1\}$. Since in G there are no cycles, such component exists if and only if the addition of the edge $(n,1)$ introduces a cycle, which happens exactly in the case when n is reachable from 1. Note that it is exactly the case when $1x = 1$ for some word $x \in \Sigma^+$.

What remains is to show that the *reduced* form $\mathbb{B}$ of $\mathbb{A}$ admits P_g if and only if $\mathbb{A}$ does. First, both 1 and $n+1$ are in the connected part $\mathbb{A}'$ of $\mathbb{A}$, and are distinguishable by the empty word (since $n+1$ is final and 1 is not). Thus, if $\mathbb{A}$ admits P_g with $1x = 1$ and $(n+1)x = n+1$ for some $x \in \Sigma^+$, then $\mathbb{B}$ admits P_g with $h(1)x = h(1)$ and $h(n+1)x = h(n+1)$ (with h being the homomorphism from the connected part of $\mathbb{A}$ onto its reduced form). For the other direction, assume $h(p)x_0 = h(p)$ for some state $p \neq n+1$ (note that since $n+1$ is the only final state, $p \neq n+1$ if and only if $h(p) \neq h(n+1)$). Let us define the sequence $p_0, p_1, \ldots$ of states of $\mathbb{A}$ as $p_0 = p$, $p_{t+1} = p_t x_0$. Then, for each $i \geq 0$, $h(p_i) = h(p)$, thus $p_i \in [n]$. Thus, there exist indices $0 \leq i < j$ with $p_i = p_j$, yielding $p_i x_0^{j-i} = p_i$, thus $\mathbb{A}$ admits P_g with $p = p_i$, $q = n+1$, $x = x_0^{j-i}$ and $y = n$.

Hence, the above construction is indeed a logspace reduction from DAG-REACH to the complement of GENDEF, showing **NL**-hardness of the latter; applying **NL** = co**NL** again, we get **NL**-hardness of GENDEF itself. □

It is worth observing that the same construction also shows **NL**-hardness (thus completeness) of the problem whether the input automaton accepts a definite language.

Thus, the complexity of the problem is characterized from the theoretic point of view. However, nondeterministic algorithms are not that useful in practice. Since **NL** $\subseteq$ **P**, the problem is solvable in polynomial time – now we give an efficient (quadratic) deterministic decision algorithm:

1. Compute $\mathbb{A}' = (Q, \Sigma, \delta, q_0, F)$, the reduced form of the input automaton $\mathbb{A}$.
2. Compute $\Gamma(\mathbb{A}')$, the çomponent graph of $\mathbb{A}'$.
3. If there exists a nontrivial, non-sink component, reject the input.
4. Compute $\mathbb{B} = \mathbb{A}' \times \mathbb{A}'$ and $\Gamma(\mathbb{B})$.
5. Check whether there exists a state (p,q) of $\mathbb{B}$ in a nontrivial component (of $\mathbb{B}$) for some $p \neq q$ with p being in the same sink as q in $\mathbb{A}$. If so, reject the input; otherwise accept it.

The correctness of the algorithm is straightforward by Theorem 2: after minimization (which takes $\mathcal{O}(n \log n)$ time) one computes the component graph of the reduced automaton (taking linear time) and checks whether there exists a nontrivial component which is not a sink (taking linear time again, since we already have the component graph). If so, then the answer is `NO`. Otherwise one has to check whether there is a (sink) component C and a word $x \in \Sigma^+$ such that $f_x|_C$ has at least two different fixed points. Now it is equivalent to ask whether there is a state (p,q) in $\mathbb{A}' \times \mathbb{A}'$ with p and q being in the same component and a word $x \in \Sigma^+$

with $(p,q)x = (p,q)$. This is further equivalent to ask whether there is a (p,q) with p,q being in the same sink such that (p,q) is in a nontrivial component of $\mathbb{B}$. Computing $\mathbb{B}$ and its components takes $\mathcal{O}(n^2)$ time, and (since we still have the component graph of $\mathbb{A}$) checking this condition takes constant time for each state (p,q) of $\mathbb{B}$, the algorithm consumes a total of $\mathcal{O}(n^2)$ time.

Hence we have a low-degree polynomial-time upper bound:

Theorem 5. *Problem* GENDEF *can be solved in* $\mathcal{O}(n^2)$ *deterministic time in the RAM model of computation.*

5 Conclusion, further directions

The forbidden pattern characterization of generalized definite languages we gave is not surprising, based on the identities of the pseudovariety of (syntactic) semigroups corresponding to this variety of languages. Still, using this characterization one can derive efficient algorithms for checking whether a given automaton recognizes such a language. Though we could not compute an exact function for the syntactic complexity, we still managed to show that these languages are not "more complex" than definite languages under this metric. Also, we gave a new upper bound for that.

The exact syntactic complexity of definite languages is still open, as well as for other language classes higher in the dot-depth hierarchy – e.g. the locally (threshold) testable and the star-free languages.

References

[1] R. S. Cohen, J. Brzozowski. Dot-Depth of Star-Free Events. Journal of Computer and System Sciences 5(1), 1971, 1–16.

[2] J. Brzozowski, D. Liu. Syntactic Complexity of Finite/Cofinite, Definite, and Reverse Definite Languages. `http://arxiv.org/abs/1203.2873`

[3] S. Cho, D. T. Huynh. The parallel complexity of finite-state automata problems. Inform. Comput. 97, 122, 1992.

[4] M. Čirič, B. Imreh, M. Steinby. Subdirectly irreducible definite, reverse definite and generalized definite automata. Publ. Electrotechn. Fak. Ser. Mat., 10, 1999, 69–79.

[5] F. Gécseg, B. Imreh. On isomorphic representations of generalized definite automata. Acta Cybernetica 15, 2001, 33–44.

[6] A. Ginzburg. About some properties of definite, reverse-definite and related automata. IEEE Trans. Electronic Computers EC-15, 1966, 809–810.

[7] M. Holzer, B. König. On deterministic finite automata and syntactic monoid size. Theoretical Computer Science 327(3), 319–347, 2004.

[8] Neil D. Jones, Y. Edmund Lien and William T. Laaser: New problems complete for nondeterministic log space. THEORY OF COMPUTING SYSTEMS Volume 10, Number 1 (1976), 1-17.

[9] O. Klíma, L. Polák. Alternative Automata Characterization of Piecewise Testable Languages. Accepted to DLT 2013.

[10] B. Krawetz, J. Lawrence, J. Shallit. State Complexity and the Monoid of Transformations of a Finite Set. Proc. of Implementation and Application of Automata, LNCS 3317, 2005, 213–224.

[11] B. Li. Syntactic Complexities of Nine Subclasses of Regular Languages. Master's Thesis.

[12] D. Perrin. Sur certains semigroupes syntactiques. Séminaires de l'IRIA, Logiques et Automates, Paris, 1971, 169–177.

[13] T. Petkovič, M. Čirič, S. Bogdanovič. Decomposition of automata and transition semigroups. Acta Cybernetica 13, 1998, 385–403.

[14] J-É. Pin. Syntactic semigroups. Chapter 10 in Handbook of Formal Languages, Vol. I, G. Rozenberg et A. Salomaa (eds.), Springer Verlag, 1997, 679–746.

[15] M. Steinby. On definite automata and related systems. Ann. Acad. Sci. Fenn., Ser. A I 444, 1969.

[16] J. Stern. Complexity of some problems from the theory of automata. Information and Control 66, 1985, 163–176.

[17] A. N. Trahtman. Piecewise and local threshold testability of DFA. Proc. of FCT 2001, LNCS 2038 (2001), 347–358.

Received 19th July 2014

Acta Cybernetica **22** (2016) 703–713.

On Chomsky Hierarchy of Palindromic Languages*

Pál Dömösi†, Szilárd Fazekas‡ and Masami Ito§

Abstract

The characterization of the structure of palindromic regular and palindromic context-free languages is described by S. Horváth, J. Karhumäki, and J. Kleijn in 1987. In this paper alternative proofs are given for these characterizations.

Keywords: palindromic formal languages, combinatorics of words and languages

1 Introduction

The study of combinatorial properties of words is a well established field and its results show up in a variety of contexts in computer science and related disciplines. In particular, formal language theory has a rich connection with combinatorics on words, even at the most basic level. Consider, for example, the various pumping lemmata for the different language classes of the Chomsky hierarchy, where applicability of said lemmata boils down in most cases to showing that the resulting words, which are rich in repetitions, cannot be elements of a certain language. After repetitions, the most studied special words are arguably the palindromes. These are sequences, which are equal to their mirror image. Apart from their combinatorial appeal, palindromes come up frequently in the context of algorithms for DNA sequences or when studying string operations inspired by biological processes, e.g., hairpin completion [2], palindromic completion [10], pseudopalindromic completion [3], etc. Said string operations are often considered as language generating formalisms, either by applying them to all words in a given language or by applying them iteratively to words. One of the main questions, when considering the languages arising from these operations, is how they relate to the classes defined by the Chomsky hierarchy. In order to investigate that, one usually needs to refer

*The second author was supported by Akita University, Dept. of Information Science and Engineering

†Institute of Mathematics and Informatics, College of Nyíregyháza, H-4400 Nyíregyháza, Sóstói út 31/B, Hungary, E-mail: domosi@nyf.hu

‡Department of Information Science and Engineering, Akita University, Akita, Tegatagakuen City 1-1, 010-8502, Japan, E-mail: szilard.fazekas@gmail.com

§Department of Mathematics, Kyoto Sangyo University, Kyoto 603, Japan E-mail: ito@cc.kyoto-su.ac.jp

DOI: 10.14232/actacyb.22.3.2016.10

to the characterization of palindromic languages, i.e., languages in which all words are palindromes.

Characterization of palindromic regular and context-free languages was given in [7]. Regular palindromic languages have a simple characterization, which is the basis (essentially using the same idea) of the characterizations of pseudopalindromic and k-palindromic languages and the decidability results rooted in them [3].

In this paper we give alternative proofs of these characterizations. Due to the previously mentioned resurgence of interest in (pseudo-)palindromic languages, we think that it is important to have clear and, where possible, effective proofs for these results readily available. The paper by Horváth et al. is correct, and it conveys the main idea characterizing palindromic languages. However, the proofs omit several (tedious) details and explicit constructions. The latter and the fact that the availability of the paper is unfortunately rather limited, are the two main reasons which prompted us to write the present work. While our line of thought is similar to the original work of Horváth et al., we make use of results discovered since then (e.g. about bounded languages) to make the proofs simpler yet complete with details. We also present some explicit constructions in the proofs, which lead to a normal form of context-free grammars generating palindromic languages. As the proofs progress, we will point out differences between our work and the arguments in [7].

2 Preliminaries

A *word* (over Σ) is a finite sequence of elements of some finite non-empty set Σ. We call the set Σ an *alphabet,* the elements of Σ *letters.* If u and v are words over an alphabet Σ, then their *catenation* uv is also a word over Σ. In particular, for every word u over Σ, $u\lambda = \lambda u = u$, where λ denotes the *empty word.* Two words u, v are said to be *conjugates* if there exists a word w with $uw = wv$. For a word w, we define the powers of w inductively, $w^0 = \lambda$ and $w^n = w^{n-1}w$, where w^n is the n-th *power* of w. A nonempty word w is called *primitive* if it is not a power of another word, i.e., $w = v^k$ implies $v = w$ and $k = 1$. Otherwise we call it a *nonprimitive word.* Thus λ is also considered a nonprimitive word.

The *length* $|w|$ of a word w is the number of letters in w, where each letter is counted as many times as it occurs. Thus $|\lambda| = 0$. By the *free monoid* Σ^* *generated by* Σ we mean the set of all words (including the *empty word* λ) having catenation as multiplication. We set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$, where the subsemigroup Σ^+ of Σ^* is said to be the *free semigroup generated by* Σ. Subsets of Σ^* are referred to as *languages* over Σ. Denote by $|H|$ the *cardinality* of H for every set H. A language L is said to be *slender* if there exists a nonnegative integer c, such that for all integers $n \geq 0$ we have $|\{w \in L : |w| = n\}| \leq c$.

For a nonempty word $w = x_1 \cdots x_n$, where $x_1, \ldots, x_n \in \Sigma$, we denote its *reverse,* $x_n \cdots x_1$, by w^R. Moreover, by definition, let $\lambda = \lambda^R$, where λ denotes the empty word of Σ^*. We say that a word w is a *palindrome* (or *palindromic*) if $w = w^R$. Further, we call a language $L \subseteq \Sigma^*$ *palindromic* if all of its elements are palindromes.

A language $L \subseteq \Sigma^*$ is called a *paired loop language* if it is of the form $L = \{uv^n wx^n y | n \geq 0\}$ for some words $u, v, w, x, y \in \Sigma^*$.

Finally, as usual, we write a *generative grammar* G into the form $G = (V, \Sigma, S, P)$, where V and Σ are disjoint nonempty finite sets, the *set of nonterminals*, and *the set of terminals*, $S \in V$ is the *start symbol*, and $P \subset (V \cup \Sigma)^* V V \times (V \cup \Sigma)^*$ is the finite set of *derivation rules*. For every *sentential form* $W \in (V \cup \Sigma)^*$, $L_G(W)$ denotes the *language generated by* W, and $L(G)$ $(= L_G(S))$ denotes the language *generated by* G. Our results are related to well-known classes of the Chomsky hierarchy, that of context-free languages and regular languages. Apart from those two, we will use the notion of *linear grammars* (languages). For all three classes, $P \subset V \times \alpha$, where $\alpha = (V \cup \Sigma)^*$ for context-free grammars, $\alpha = \Sigma^*(V \cup \{\lambda\})\Sigma^*$ for linear grammars, and $\alpha = \Sigma^*(V \cup \{\lambda\})$ for regular grammars.

We shall use the following classical results.

Theorem 1. *[1] Let L be a regular language. Then there is a constant n such that if z is any word in L, and $|z| \geq n$, we may write $z = uvw$ in such a way that $|uv| \leq n, |v| \geq 1$, and for all $i \geq 0$, $uv^i w$ is in L. Furthermore, n is no greater than the number of states of the finite automaton with minimal states accepting L.*

Theorem 2. *The family of context-free languages is closed under the inverse homomorphism.*

Theorem 3. *[1] The language $L \subseteq \Sigma^*$ is context-free if and only if for every regular language $R \subseteq \Sigma^*$, $L \cap R$ is context-free.*

Theorem 4. *[6] Given an alphabet Σ, a nonempty word $w \in \Sigma^+$, each context-free language $L \subseteq w^*$ is regular having the form*

$$\cup_{i=1}^{k} w^{m_i}(w^{n_i})^* \text{ for some } m_1, n_1, \ldots, m_k, n_k \geq 0. \tag{1}$$

Theorem 5. *[8, 9, 12] Every slender context-free language is a finite disjoint union of paired loop languages.*

The following statement is well-known.

Proposition 1. *Given a context-free grammar $G = (V, \Sigma, S, P)$, a sentential form $W \in (V \cup \Sigma)^*$, the language $S_G(W)$ is also context-free.*

Theorem 6. *[13] Given a positive integer i, a pair $u, v \in \Sigma^+$, let $uv = p^i$ for some primitive word $p \in \Sigma^+$. Then $vu = q^i$ for a primitive word q.*

Theorem 7. *[11] If $uv = vq, u \in \Sigma^+, v, q \in \Sigma^*$, then $u = wz, v = (wz)^k w, q = zw$ for some $w \in \Sigma^*, z \in \Sigma^+$ and $k \geq 0$.*

Theorem 8. *[11] The words $u, v \in \Sigma^*$ are conjugates if and only if there are words $p, q \in \Sigma^*$ with $u = pq$ and $v = qp$.*

Theorem 9. *[4] Let $u, v \in \Sigma^*$. $u, v \in w^+$ for some $w \in \Sigma^+$ if and only if there are $i, j \geq 0$ so that u^i and v^j have a common prefix (suffix) of length $|u| + |v| - gcd(|u|, |v|)$.*

We shall use the following direct consequence of this result.

Theorem 10. *If two non-empty words p^i and q^j share a prefix of length $|p| + |q|$, then there exists a word r such that $p, q \in r^+$.*

3 Results

We start with alternative proofs of some results of S. Horváth, J. Karhumäki, J. Kleijn [7].

First we turn to consider regular languages. We present a proof which is shorter than the one in [7] and does not make direct reference to the underlying finite automata and is instead based solely on the pumping lemma for regular languages and combinatorial results. The following is a simple result, and essentially the same idea has been used for instance for the characterization of pseudopalindromic regular languages [3].

Theorem 11. *[7] A regular language $L \subseteq \Sigma^*$ is palindromic if and only if it is a union of finitely many languages of the form*

$$L_p = \{p\}, L_{q,r,s} = qr(sr)^*q^R, (p, q, r, s \in \Sigma^*), \tag{2}$$

where p, r and s are palindromes.

Proof. Clearly, any finite union of languages in (2) is both palindromic and regular. Conversely, let L be a palindromic regular language and n be the language-specific constant from Theorem 1. Naturally, there are finitely many words shorter than n, those will form the languages L_p. For any suitably long word $w \in L$, according to Theorem 1, we have a factorization $w = qvz$, with $0 < |qv| \leq n$ and $v \neq \lambda$, such that $qv^iz \in L$, for any $i \geq 0$. The two cases being symmetric, we may assume $|q| \leq |z|$, i.e., $z = xq^R$, for some $x \in \Sigma^*$, with v^ix being a palindrome. This gives us $x = r(v^R)^j$, for some r with $v^R = sr$ and some $j \geq 0$. But, for large enough i, v^ix ends in $sx = (v^Rv^R)^Rx = (r^Rs^R)^2r(v^R)^j$ and it starts with v^{j+2}, so we instantly get $v = r^Rs$ and thus $s = s^R$. It also follows, that $v^R = s^Rr$ and $v^R = s^Rr^R$, hence r is a palindrome, too. Then, our original word w can be written as $qr(sr)^{j+k}q^R$. A similar decomposition, according to Theorem 1 is bound to exist for all words longer than n. All parts of the decomposition, q, r and s are shorter than n, therefore there are finitely many triplets like this.

□

Next we prove the following simple observation.

Proposition 2. *Given a pair of positive integers i, j, let $p, r, u, w \in \Sigma^*, v \in \Sigma^+$ be arbitrary with $|p| \leq |u|, |r| \leq |w|$ and let $q \in \Sigma^+$ be a primitive word having $|v^j| \geq |v| + 3|q|$ such that $pq^ir = uv^jw$. Then there exists a positive integer k such that v and q^k conjugate.*

Proof. By our assumptions, there exists a pair of factorizations $u = pu'$, $w = v'q$ such that $q^i = u'v^jv'$. Because $|v^j| \geq |v| + 3|q|$, $|u'v'| = |q^i| - |v^j| \leq |q^i| - |v| - 3|q| < |q^{i-3}|$, there are a positive integer n, a suffix q_2 and a prefix q_3 of q such that $v^j = q_2q^nq_3$. Hence $v^j = q_2(q_1q_2)^nq_3 = (q_2q_1)^nq_2q_3$ for some decomposition $q = q_1q_2$ and prefix q_3 of q. By our conditions, $|v^j| - |q_3| \geq |v| + 3|q| - |q_3| \geq |v| + 2|q| > |v| + |q|$. Therefore, applying Theorem 10, we obtain $v, q_2q_1 \in z^+$ for some primitive word $z \in \Sigma^+$. By Theorem 6, q_2q_1 is also primitive. Therefore, $z = q_2q_1$. Hence $v = (q_2q_1)^k$ for some $k > 0$. Then Theorem 8 implies that v and q^k conjugate. □

Now we continue with palindromic context-free languages. The line of thought is similar to the one in [7]. The main differences are as follows. The original proof of Theorem 12 is very succinct and only hints at the constructions needed to transform context-free grammars generating palindromic languages into linear grammars. We develop the result in detail. Afterwards, we show that for a linear grammar generating a palindromic language, one can find a "normal form", called palindromic grammar in [7]. Again, the original proof provides the combinatorial arguments to show that this is possible, but does not give an explicit construction. We present such a construction in the proofs of Lemmas 4 and 5. The technical details might at times be somewhat difficult to follow due to the proliferation of notation. To remedy that as much as possible, we decomposed the proofs in several lemmas.

Lemma 1. *Let $G = (V, \Sigma, S, P)$ be a context-free grammar, such that $L(G)$ is palindromic. Then, for any rule of the form $X \to pAqBr \in P$, with $p, q, r \in \Sigma^*$, $X, A, B \in V$, and $|L_G(A)| > 1$, $|L_G(B)| > 1$, we have that both $L_G(A)$ and $L_G(B)$ are slender context-free languages.*

Proof. Without loss of generality we can assume that V is reduced, i.e., for every $X \in V$, $L_G(X) \neq \emptyset$.

We will show that for every $q_1, q_2 \in \Sigma^*$, with $A \overset{*}{\underset{G}{\Rightarrow}} q_1, A \overset{*}{\underset{G}{\Rightarrow}} q_2$, we have that $q_1 \neq q_2$ implies $|q_1| \neq |q_2|$. Similarly, for every $r_1, r_2 \in \Sigma^*$, with $B \overset{*}{\underset{G}{\Rightarrow}} r_1, B \overset{*}{\underset{G}{\Rightarrow}} r_2$, we have $r_1 \neq r_2$ implies $|r_1| \neq |r_2|$. Because G is reduced, there are $u, y \in \Sigma^*$ having $S \overset{*}{\underset{G}{\Rightarrow}} uXy$. Therefore, $A \overset{*}{\underset{G}{\Rightarrow}} q_1$ and $A \overset{*}{\underset{G}{\Rightarrow}} q_2$ imply that for every $r' \in L_G(B)$, $upq_1qr'ry, upq_2qr'ry \in L(G)$, i.e., both of them are palindromes. This is impossible if $|q_1| = |q_2|$ with $q_1 \neq q_2$, unless $q_1 = xz_1x'$ and $q_2 = x''z_2x'''$, where z_1 and z_2 are palindromes and $upx = (x'qr'ry)^R, upx'' = (x'''qr'ry)^R$. However, then for any $r'' \in L_G(B)$ different from r', one of the words $upq_1qr''ry, upq_2qr''ry$ will not be a palindrome, but should be in $L(G)$, a contradiction.

Similarly, $B \overset{*}{\underset{G}{\Rightarrow}} r_1$ and $B \overset{*}{\underset{G}{\Rightarrow}} r_2$ imply that for every $q' \in L_G(A)$, we have $upq'qr_1ry$, $upq'qr_2ry \in L(G)$, i.e., both of them are palindromes. This is impossible if $|r_1| = |r_2|$ and $r_1 \neq r_2$, and $|L_G(A)| > 1$. This means, that both $L_G(A)$ and $L_G(B)$ are slender context-free. □

Lemma 2. *Let L_1 and L_2 be paired loop languages. If L_1L_2 is palindromic, then L_1L_2 can be generated by a linear grammar.*

Proof. The words in L_1L_2 are of the form $u_1v_1^iw_1x_1^iu_2v_2^jw_2x_2^ju_3$ and we assume they are palindromes for any $i, j \geq 0$.

If one of the words v_1, x_1, v_2, x_2 is empty, then we can generate L_1L_2 with linear rules, e.g., if x_1 is empty then we can generate $u_1v_1^iw_1$, $i \geq 0$, by linear rules $X \to u_1A$, $A \to v_1A$, $A \to w_1u_2B$ and the rest of the word by linear rules $B \to Cu_3$, $C \to v_2Cx_2$, $C \to w_2$.

Therefore, if one of v_1, x_1, v_2, x_2 is empty then we are ready, so let us assume that none of them are λ.

W.l.o.g. we may assume that $|u_1| \geq |u_3|$. Choose $j \geq 2$ such that:

- $|x_2^ju_3| - |u_1| \leq 2|x_2|$,
- $|u_1v_1^2| \leq |x_2^ju_3|$ and
- $|v_2^j| \geq 2|v_1|$.

Choose i such that $|u_1v_1^i| \geq |u_2v_2^jw_2x_2^ju_3|$. As the word is a palindrome, this means that $(u_2v_2^jw_2x_2^ju_3)^Rt = u_1v_1^i$, for some possibly empty word t. By Theorem 9, we get that the primitive roots of v_1, v_2^R, x_2^R are all conjugates of some primitive word z and $(u_2v_2^jw_2x_2)^R$ is a factor of z^k, for large enough k. If we choose j and i such that $|v_2^ju_3| > |u_1v_1^iw_1x_1^i|$ and $|x_1^i| > 2|x_2|$, then again from Theorem 9, we get that the primitive root of x_1 is also a conjugate of z. Moreover, if we choose i such that either v_1 or x_1 is in the middle of the word, then we get that there exist some palindromes z_1, z_2 such that z_1z_2 is a conjugate of z. This means that for any i, j we have $u_1v_1^iw_1x_1^iu_2v_2^jw_2x_2^ju_3 \in u_3^R(z_1z_2)^+z_1u_3$. As $|v_1|$, $|x_1|$, $|v_2|$ and $|x_2|$ are all multiples of $|z_1z_2|$, we get that L can be generated by a linear grammar with derivation rules of the form $S \to u_3^Rz_1Xu_3$ and $X \to (z_2z_1)^{n_1}X$, $X \to (z_2z_1)^{n_2}X$, $X \to (z_2z_1)^m$, for some positive integers m, n_1, n_2, such that $n_1 \cdot |z| = |v_1x_1|$, $n_2 \cdot |z| = |v_2x_2|$ and $m \cdot |z| = |w_1| + |u_2| + |w_2| + (|u_1| - |u_3| - |z_1|)$. □

Theorem 12. *[7] Every palindromic context-free language is linear.*

Proof. Let $G = (V, \Sigma, S, P)$ be a context-free grammar generating the palindromic language L. Without loss of generality we can assume that V is reduced, i.e., for every $X \in V$, $L_G(X) \neq \emptyset$. In particular, we may assume for every $X \in V$, $|L_G(X)| = \infty$. Indeed, if $|L_G(X)| < \infty$, then we can eliminate the derivation rules

$$Y \to W_1XW_2X \cdots W_nXW_{n+1}, X \to W \in P,$$

$W, W_1, W_2, \ldots, W_{n+1} \in ((V \setminus \{X\}) \cup \Sigma)^*$ by new derivation rules of the form

$$Y \to W_1w_1W_2w_2 \cdots w_nW_{n+1}, w_1, \ldots, w_n \in L_G(X).$$

It can also be assumed that for every $X \to W \in P$, there are at most two (not necessarily different) nonterminals appearing in W. Indeed, if

$X \to u_1 A_1 \cdots u_n A_n u_{n+1} \in P$ with $X, A_1, \ldots, A_n \in V, u_1, \ldots, u_n \in \Sigma^*, n > 2$ then we can eliminate this derivation rule by the following new derivation rules using some new nonterminals $A'_1, \ldots, A'_{n-1}$:

$$X \to u_1 A_1 u_2 A'_2, A'_2 \to A_2 u_3 A'_3, \ldots, A'_{n-2} \to A_{n-2} u_{n-1} A'_{n-1}, A'_{n-1} \to A_{n-1} u_n.$$

Next we show that the derivation rules of the form $X \to pAqBr$ with $p, q, r \in \Sigma^*, A, B \in V$ can be eliminated.

Since we assumed $L_G(A)$ and $L_G(B)$ are infinite languages, by Lemma 1 both of them are slender context-free languages, hence so are $\{p\} \cdot L_G(A) \cdot \{q\}$ and $L_G(B) \cdot \{r\}$. Using Theorem 5, we get that $L_G(pAqBr)$ is a concatenation of two paired loop languages and it is palindromic. From here, applying Lemma 2 gives that $L_G(pAqBr)$ can be generated by linear derivation rules.

Thus we receive that $L(G)$ can be generated by a linear grammar. □

Lemma 3. *Given an alphabet Σ, words $v, z \in \Sigma^*$, a non-empty word $w \in \Sigma^+$, each context-free language $L \subseteq vw^*z$ is regular having the form*

$$v(\cup_{i=1}^{k} w^{m_i}(w^{n_i})^*)z \text{ for some } m_1, n_1, \ldots, m_k, n_k \geq 0. \tag{3}$$

Proof. Let a, b, c distinct symbols and consider a homomorphism $\psi : \{a, b, c\} \to \Sigma^*$ with $\psi(a) = v, \psi(b) = w, \psi(c) = z$. Then $\psi^{-1}(L) \cap ab^*c = \{ab^kc \mid vw^kz \in L, k \geq 0\}$. On the other hand, using that ab^*c is obviously a regular language, Theorem 2 and Theorem 3 imply that $\psi^{-1}(L) \cap ab^*c$ is also context-free. Let $\psi' : \{a, b, c\} \to b^*$ be a homomorphism with $\psi'(a) = \psi'(c) = \lambda$ and $\psi'(b) = b$. By Theorem 2, $\psi'(\psi^{-1}(L) \cap ab^*c)$ is also context-free. On the other hand, $\psi'(\psi^{-1}(L) \cap ab^*c) = \{b^k \mid vw^kz \in L, k \geq 0\}$, therefore, by Theorem 4, it is regular which can be written into the form $\cup_{i=1}^{k} b^{m_i}(b^{n_i})^*$ for some $m_1, n_1, \ldots, m_k, n_k \geq 0$. This implies that L is regular having the form as in (3). □

Given a grammar $G = (V, \Sigma, S, P)$, we say that a nonterminal $X \in V$ is *non-balanced* if there are $p, q \in \Sigma^*$ with $|p| \neq |q|$ such that $X \overset{*}{\underset{G}{\Rightarrow}} pXq$. Otherwise, we say that X is *balanced*. We will show that for each palindromic context-free language, there exists a linear grammar in a palindromic normal form. The proof requires two steps: first we show that such languages can be generated by grammars with balanced nonterminals, and then we show that any grammar with balanced nonterminals can be effectively transformed into a grammar in palindromic normal form.

Lemma 4. *Every palindromic context-free language can be generated by a $G = (V, \Sigma, S, P)$, such that each non-terminal in V is balanced.*

Proof. Consider an arbitrary palindromic context-free language L. By Theorem 12, we have that L is linear. Thus there exists a linear grammar $G = (V, \Sigma, S, P)$, such that $L(G) = L$. Without loss of generality, we may assume that G is reduced, moreover, $P \subseteq \{X \to aYb \mid X \in V, Y \in V \cup \{\lambda\}, a, b \in \Sigma \cup \{\lambda\}, ab \neq \lambda\}$. Indeed, if $X \to paYbq \in P$ with $p, q \in \Sigma^*, pq \in \Sigma^+, a, b \in \Sigma \cup \{\lambda\}, ab \neq \lambda, Y \in V \cup \{\lambda\}$,

then we can eliminate the derivation rule $X \to paYbq \in P$ by introducing a new nonterminal symbol Z and the new derivation rules $X \to pZq, Z \to aYb$. Thus we get in finite-many steps that all derivation rules have the form $X \to aYb, X \in V, a, b \in \Sigma \cup \{\lambda\}, Y \in V \cup \{\lambda\}$.

Clearly, then

$$L = \cup\{\{p\}L_G(X)\{q\} \mid S \overset{*}{\underset{G}{\Rightarrow}} pXq,\ X \in V, p, q \in \Sigma^*, |p|, |q| \leq |V|\}. \tag{4}$$

Consider a non-balanced nonterminal X, as above. Let us assume X appears in a derivation at some point as $S \Rightarrow uXv$. Then, because $X \Rightarrow pXq$, we get $S \Rightarrow up^iXq^iv$, for all $i \geq 1$. Without loss of generality, we may assume $|u| \leq |v|$, that is, since the derived word will be a palindrome, $v = wu^R$, for some $w \in \Sigma^*$. Now, to keep arguments simple, let X stand for any word in $L_G(X)$. So, we know that p^iXq^iw is a palindrome for any positive i. For large enough i, this gives us that $w^R = p^jp_1$, for some $j \geq 0$ and $p_1 \in \Sigma^*$ prefix of p, hence $p^iXq^ip_1^R(p^R)^j$ is a palindrome. Again, if i was big enough for $|p^i| > |q^2p_1^R(p^R)^j|$, then by Theorem 9, we get that for a decomposition q_1q_2 of q^R, its conjugate q_2q_1 has the same primitive root as p, i.e., there exists some primitive word $z \in \Sigma^+$, $m, n \geq 1$, such that $q_2q_1 = z^m$ and $p = z^n$. Rewriting $p^iXq^ip_1^R(p^R)^j$ with these powers of z, we have $z^{ni}X(q_2^Rq_1^R)^ip_1(z^R)^{nj} = z^{ni}Xq_2^R(q_1^Rq_2^R)^{i-1}q_1^Rp_1(z^R)^{nj} = z^{ni}Xq_2^R(z^R)^{m(i-1)}q_1^Rp_1(z^R)^{nj}$ is a palindrome, therefore $z^{n(i-j)}Xq_2^R(z^R)^{m(i-1)}q_1^Rp_1$ is, as well. This means $p_1^Rq_1z^2$ is a prefix of $z^{n(i-j)}$, and we can apply Theorem 9 again to get that, since z is primitive, $p_1^Rq_1 = z^k$, for some integer k. Since p_1^R is a suffix of $p^R = (z^R)^n$ and q_1 is a suffix of z^m, there exist non-negative integers i_1, i_2 and z_r' suffix of z^R, z' suffix of z, such that $z_r'(z^R)^{i_1}z'z^{i_2} = z^k$. From here, there is some prefix z_r'' of z^R, with $z_r''z_r' = z^R$, $z_r'z_r'' = z$, so both z_r'' and z_r' are palindromes and so are $p_1 = z_r'(z_r''z_r')^{i_1}$ and $q_1 = (z_r''z_r')^{k-i_1-1}z_r''$. But $q_2q_1 = z^m = (z_r'z_r'')^m$, so $q_2 = z_r'(z_r''z_r')^{m-k+i_1+1}$. From here, $z^{ni}X(q_2^Rq_1^R)^ip_1(z^R)^{nj} = (z_r'z_r'')^{ni}X(z_r'z_r'')^{mi}z_r'(z_r''z_r')^{i_1}(z_r''z_r')^{nj} = (z_r'z_r'')^{ni}X(z_r'z_r'')^{mi+i_1+nj}z_r'$ is a palindrome for all $i \geq 1$. As our original assumption was $|p| \neq |q|$, i.e., $m \neq n$, for a large enough i, the word X will be entirely to the left or right from the center of a palindrome of the form $(z_r'z_r'')^{j_1}X(z_r'z_r'')^{j_2}z_r'$. Since $z_r'z_r''$ is primitive, the center of the palindrome has to be exactly z_r' or z_r'', and this means that $X \in (z_r'z_r'')^+$. Then, the language $L_G(X)$ is isomorphic to a unary context-free language, hence it is regular with rules of the form $X \to (z_r'z_r'')^{m+n}X$. This way, in our original grammar we can replace all rules with X on the left with balanced rules $X \to (z_r'z_r'')^{\frac{m+n}{2}}X(z_r'z_r'')^{\frac{m+n}{2}}$ and $X \to \lambda$, or if $m+n$ is odd, with rules $X \to (z_r'z_r'')^{m+n}X(z_r'z_r'')^{m+n}$ and $X \to (z_r'z_r'')^{m+n}|\lambda$.

□

Lemma 5. *Every palindromic context-free language can be generated by a grammar* $G = (V, \Sigma, S, P)$ *having* $P \subseteq \{X \to aYa \mid X, Y \in V, a \in \Sigma\} \cup \{X \to a \mid X \in V, a \in \Sigma\} \cup \{X \to \lambda\}$.

Proof. Now we may assume that V contains only balanced nonterminals, i.e., for every derivation, $X \overset{*}{\underset{G}{\Rightarrow}} uXx$, where $X \in V$, $u, x \in \Sigma^*$, $|u| = |x|$. Then, for every

$X \in V$, $p, q \in \Sigma^*$, $S \overset{*}{\underset{G}{\Rightarrow}} pXq$ implies $||p| - |q|| < |V|$. This obviously holds for derivations of less than $|V|$ steps, as in each step we add at most one letter to either side. Assume the contrary for a longer derivation:

$$X_0 \underset{G}{\Rightarrow} x_1 X_1 y_1 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} x_{n-1} X_{n-1} y_{n-1} \cdots y_1 \underset{G}{\Rightarrow} x_1 \cdots x_n X_n y_n \cdots y_1, \tag{5}$$

where $X_0 = S$, $x_1, \ldots, x_n, y_1, \ldots, y_n \in \Sigma \cup \{\lambda\}$ and $n > |V|$. Then, there exist $0 \leq i < j \leq n$, such that $X_i = X_j$, but X_i is balanced, so $|x_i \cdots x_j| = |y_j \cdots y_i|$, therefore we can remove them from both sides and get that $||x_1 \cdots x_n| - |y_n \cdots y_1|| = ||x_1 \cdots x_{i-1} x_{j+1} \cdots x_n| - |y_n \cdots y_{j+1} y_{j-1} \cdots y_{i+1}||$. Repeating this until we get a derivation with at most $|V|$ steps, gives us $||x_1 \cdots x_n| - |y_n \cdots y_1|| \leq |V|$.

Now, to every derivation, we assign two queues (first-in-first-out storages), called *left store* and *right store*. Either both of them are empty, or one of them is empty and the other one contains a non-empty terminal string of length less than $|V|$.

At the start, both stores are empty. This status does not change as long as the applied derivation rules are of the form $X \to aYa$, $X, Y \in V, a \in \Sigma \cup \{\lambda\}$. If the applied derivation rule has the form $X \to aY, X, Y \in V, a \in \Sigma$, then there are two cases: if the left store is empty, then we drop the terminal letter a onto the top of the right store; otherwise we delete the terminal letter contained at the bottom of the left store. In the second case, the bottom of the left store should contain the same terminal letter a. Otherwise the generated word will not be a palindrome. Similarly, if the applied derivation rule has the form $X \to Yb, X, Y \in V, b \in \Sigma$, then we have two cases: if the right store is empty, then we drop the terminal letter b onto the top of the left store; otherwise we delete the terminal letter contained at the bottom of the right store. In the second case again, the bottom of the right store should contain the same terminal letter b. Otherwise the generated word will not be a palindrome.

If the applied derivation rule has the form $X \to aYb, X, Y \in V, a, b \in \Sigma$, then we have the following possibilities: if one of the stores is not empty, then our procedure works as in the previous cases (like, in order, applying a derivation rule $X \to aZ, a \in \Sigma, X, Z \in V$, and then a derivation rule $Z \to Yb, b \in \Sigma, Z, Y \in V$); if both stores are empty then $a = b$ should hold, otherwise the generated string will not be a palindrome. After applying the considered derivation rule $X \to aYb, X, Y \in V, a, b \in \Sigma$, the contents of the stores remain the same.

We will construct our grammar such that a derivation rule of the form $X \to a, a \in \Sigma \cup \{\lambda\}, X \in V$ can be applied only if either one of the stores contains the letter a or both stores are empty.

In addition, if both stores are empty, and $X \overset{*}{\underset{G}{\Rightarrow}} w$ may hold for the nonterminal X contained on the left-hand side of the applied derivation rule, then w should be a palindrome. In addition, if $|w| < |V|$, then either $w = b$ with $b \in \Sigma \cup \{\lambda\}$, or $w = c_1 \cdots c_t d c_t \cdots c_1$ for some $c_1, \ldots, c_t \in \Sigma, d \in \Sigma \cup \{\lambda\}, 1 \leq t < |V|$. For the second case, we assume the existence of some derivation rules of the form $X \to c_1 Z_1 c_1, Z_1 \to c_2 Z_2 c_2, \ldots, Z_{t-1} \to c_t Z_t c_t, Z_t \to d, Z_1, \ldots, Z_t \in V$.

Having these properties, we formally define the following set of derivation rules, where the (new) nonterminals are supplied by the queues discussed above.

Let $\bar{V} = \{X \in V \mid X \overset{*}{\underset{G}{\Rightarrow}} w, w \in \Sigma^+, |w| < |V|\}$ and define, in order,
$V' = \{X_{\lambda,\lambda} \mid X \in V\} \cup \{X_{a_1 \cdots a_k,\lambda} \mid X \in V, a_1, \ldots, a_k \in \Sigma, k < |V|\}$
$\cup \{X_{\lambda,b_1 \cdots b_k} \mid X \in V, b_1, \ldots, b_k \in \Sigma, k < |V|\}$
and
$P' = \{X_{a_1 \cdots a_k,\lambda} \to aY_{a_1 \cdots a_k a,\lambda}a, X_{\lambda,a_1 \cdots a_k} \to Y_{\lambda,a_1 \cdots a_{k-1}}, X_{\lambda,\lambda} \to aY_{a,\lambda}a$
$\mid X \to Ya \in P, X, Y \in V, a_1, \ldots, a_k, a \in \Sigma, k < |V|\} \cup$
$\{X_{a_1 \cdots a_k,\lambda} \to Y_{a_1 \cdots a_{k-1},\lambda}, X_{\lambda,a_1 \cdots a_k} \to aY_{\lambda,a_1 \cdots a_k a}a, X_{\lambda,\lambda} \to aY_{\lambda,a}a$
$\mid X \to aY \in P, X, Y \in V, a_1, \ldots, a_k, a \in \Sigma, k < |V|\} \cup$
$\{X_{a_1 \cdots a_k,\lambda} \to bY_{a_1 \cdots a_{k-1}b,\lambda}b, X_{\lambda,a_1 \cdots a_k} \to aY_{\lambda,a_1 \cdots a_{k-1}a}a, X_{\lambda,\lambda} \to aY_{\lambda,\lambda}b$
$\mid X \to aYb \in P, X, Y \in V, a_1, \ldots, a_k, a, b \in \Sigma \cup \{\lambda\}\} \cup$
$\{X_{a_1 \cdots a_k,\lambda} \to Y_{a_1 \cdots a_k,\lambda}, X_{\lambda,a_1 \cdots a_k} \to Y_{\lambda,a_1 \cdots a_k}, X_{\lambda,\lambda} \to Y_{\lambda,\lambda}$
$\mid X \to Y \in P, X, Y \in V, a_1, \ldots, a_k, \in \Sigma \cup \{\lambda\}\} \cup \{X_{a,\lambda} \to \lambda, X_{\lambda,a} \to \lambda,$
$X_{\lambda,\lambda} \to a \mid X \to a \in P, X \in V, a \in \Sigma\} \cup$
$\{X_{\lambda,\lambda} \to \lambda \mid X \to \lambda \in P\} \cup \{X_{\lambda,\lambda} \to c_1 Z_{1X\lambda,\lambda}c_1,$
$Z_{1X\lambda,\lambda} \to c_2 Z_{2X\lambda,\lambda}c_2, \ldots, Z_{t-1X\lambda,\lambda} \to c_t Z_{tX\lambda,\lambda}c_t, Z_{tX\lambda,\lambda} \to d \mid X \in \bar{V},$
$X \overset{*}{\underset{G}{\Rightarrow}} c_1 \cdots c_t d c_t \cdots c_1, c_1, \ldots, c_t \in \Sigma, d \in \Sigma \cup \{\lambda\}\}$.

Thus we get that $L(G) = L(G')$, where $G' = (V', \Sigma, S_{\lambda,\lambda}, P')$, and G' has the desired form. □

Theorem 13. *[7] A context-free language $L \subseteq \Sigma^*$ is palindromic if and only if it is a disjoint union of $|V|$ languages of the form $\{pap^R \mid p \in L_a\}$, where the L_a $(a \in \Sigma \cup \{\lambda\})$ are regular languages (uniquely determined by L).*

Proof. Given an alphabet Σ, for every $a \in \Sigma \cup \{\lambda\}$ consider a regular language L_a. It is clear that $L = \bigcup_{a \in \Sigma \cup \{\lambda\}} \{pap^R : p \in L_a\}$ is palindromic and linear (and thus, it is also context-free). Conversely, consider a palindromic context-free language L. By Lemma 5, it can be generated by a grammar $G = (V, \Sigma, S, P)$ having $P \subseteq \{X \to aYa \mid X, Y \in V, a \in \Sigma\} \cup \{X \to a \mid X \in V, a \in \Sigma\} \cup \{X \to \lambda \mid X \in \Sigma\}$. For every $a \in \Sigma \cup \{\lambda\}$, define the grammar $G_a = (V, \Sigma, S, P_a)$ with $P_a = P \setminus \{X \to b \mid b \in \Sigma \cup \{\lambda\}, b \neq a\})$. Obviously, $L(G) = \cup_{a \in \Sigma} L(G_a)$. Moreover. for every $a, b \in \Sigma \cup \{\lambda\}$, $L(G_a) \cap L(G_b) \neq \emptyset$ if and only if $a = b$. Therefore, L is a disjoint union of the languages $L(G_a), a \in \Sigma \cup \{\lambda\}$. By the construction of $G_a, a \in \Sigma \cup \{\lambda\}$, it is clear that $G_{a,\ell} = (V, \Sigma, S, P_{a,\ell}$ with $P_{a,\ell} = \{X \to Yb \mid X \to bYb \in P_a, X, Y \in V, a \in \Sigma\} \cup \{X \to b \mid X \to b \in P_a, X \in V, a \in \Sigma \cup \{\lambda\}\}$ is a regular language. Similarly, $G_{a,r} = (V, \Sigma, S, P_{a,r}$ with $P_{a,r} = \{X \to bY \mid X \to bYb \in P_a, X, Y \in V, a \in \Sigma\} \cup \{X \to b \mid X \to b \in P_a, X \in V, a \in \Sigma \cup \{\lambda\}\}$ is regular. Moreover, $L_a = L(G_{a,\ell}) = L(G_{a,r})$, and $L = \bigcup_{a \in \Sigma \cup \{\lambda\}} \{pap^R : p \in L_a\}$. □

Finally, for the sake of completeness, let us make an easy observation. Every palindromic context-sensitive (phrase-structured) language has the form

$$L = \bigcup_{a \in \Sigma \cup \{\lambda\}} \{pap^R : p \in L(a)\},$$

where the $L(a)$ $(a \in \Sigma \cup \{\lambda\})$ are context-sensitive (phrase-structured) languages (uniquely determined by L).

References

[1] Bar-Hillel, Y.; Perles, M.; Shamir, E.: On formal properties of simple phrase structure grammars. Zeitschrift für Phonetik, Sprachwuissenschaft, und Kommunikationsforschung, **14** (1961), 143-177.

[2] Cheptea, D; Martín-Vide, C.; Mitrana, V.: A new operation on words suggested by DNA biochemistry: Hairpin completion. *In Proc. Conf. Transgressive Computing*, 2006, 216-228.

[3] Fazekas, S.Z.; Manea, F.; Mercas, R.; Shikishima-Tsuji, K.: The pseudopalindromic completion of regular languages. Inform. Comput. **239** (2014), 222-236.

[4] Fine, N. J.; Wilf, H. S.: Uniqueness theorems for periodic functions. Proc. Am. Math. Soc. **16** (1965), 109-114.

[5] Ginsburg, S.; Spanier, E. H.: Bounded ALGOL-like languages. *Trans. Am. Math. Soc.*, **113** (1964), 333-368.

[6] Ginsburg, S.; Rice, H. G.: Two families of languages related to ALGOL. *J. Assoc. Computing Machinery*, **9** (1962), 350–371.

[7] Horváth, S.; Karhumäki, J.; Kleijn, J.: Results concerning palindromicity. (Mathematical aspects of informatics, Mägdesprung, 1986). *J. Inform. Process. Cybernet.* **23** (1987), no. 8-9, 441–451.

[8] Ilie, L.: On a conjecture about slender context-free languages. *Theoret. Comput. Sci.*, **132** (1994), 427–434.

[9] Latteux, M; Thierrin, G.: Semidiscrete context-free languages. *Internat. J. Comput. Math.* **14** (1983), 3–18.

[10] de Luca, A; Luca, A.D.: Pseudopalindrome closure operators in free monoids. *Theoret. Comput. Sci.*, **362** (2006), 282-300.

[11] Lyndon, R. C.; Schützenberger, M. P.: The equation $a^m = b^n c^p$ in a free group. *Michigan Math. J.*, **9** (1962), 289-298.

[12] Raz, D.: Length considerations in context-free languages. *Theoret. Comput. Sci.*, **183** (1997), 21–32.

[13] Shyr, H. J.; Thierrin, G.: Disjunctive languages and codes. *In: Karpiński (ed.): Proc. Conf. FCT'77*, **56** (1977), Springer-Verlag, 171–176.

Received 11th October 2013

ACTA CYBERNETICA

Information for authors. Acta Cybernetica publishes only original papers in the field of Computer Science. Manuscripts must be written in good English. Contributions are accepted for review with the understanding that the same work has not been published elsewhere. Papers previously published in conference proceedings, digests, preprints are eligible for consideration provided that the author informs the Editor at the time of submission and that the papers have undergone substantial revision. If authors have used their own previously published material as a basis for a new submission, they are required to cite the previous work(s) and very clearly indicate how the new submission offers substantively novel or different contributions beyond those of the previously published work(s). Each submission is peer-reviewed by at least two referees. The length of the review process depends on many factors such as the availability of an Editor and the time it takes to locate qualified reviewers. Usually, a review process takes 6 months to be completed. There are no page charges. An electronic version of the puplished paper is provided for the authors in PDF format.

Manuscript Formatting Requirements. All submissions must include a title page with the following elements:

- title of the paper
- author name(s) and affiliation
- name, address and email of the corresponding author
- An abstract clearly stating the nature and significance of the paper. Abstracts must not include mathematical expressions or bibliographic references.

References should appear in a separate bibliography at the end of the paper, with items in alphabetical order referred to by numerals in square brackets. Please prepare your submission as one single PostScript or PDF file including all elements of the manuscript (title page, main text, illustrations, bibliography, etc.). Manuscripts must be submitted by email as a single attachment to either the most competent Editor, the Managing Editor, or the Editor-in-Chief. In addition, your email has to contain the information appearing on the title page as plain ASCII text. When your paper is accepted for publication, you will be asked to send the complete electronic version of your manuscript to the Managing Editor. For technical reasons we can only accept files in LaTeX format.

Subscription Information. Acta Cybernetica is published by the Institute of Informatics, University of Szeged, Hungary. Each volume consists of four issues, two issues are published in a calendar year. Subscription rates for one issue are as follows: 5000 Ft within Hungary, €40 outside Hungary. Special rates for distributors and bulk orders are available upon request from the publisher. Printed issues are delivered by surface mail in Europe, and by air mail to overseas countries. Claims for missing issues are accepted within six months from the publication date. Please address all requests to:

Acta Cybernetica, Institute of Informatics, University of Szeged
P.O. Box 652, H-6701 Szeged, Hungary
Tel: +36 62 546 396, Fax: +36 62 546 397, Email: acta@inf.u-szeged.hu

Web access. The above information along with the contents of past issues are available at the Acta Cybernetica homepage https://www.inf.u-szeged.hu/en/kutatas/acta-cybernetica .

CONTENTS

ISSN 0324—721 X

Felelős szerkesztő és kiadó: Csirik János